On the basis of your performance in this examination, the examiners will provide results on the following criterion taken from the course statement:

**Criterion 1**  Design and evaluate algorithmic solutions to a range of problems.
CANDIDATE INSTRUCTIONS

Candidates MUST ensure that they have addressed the externally assessed criterion on this examination paper.

Answer ALL questions. Answers must be written in the spaces provided on the examination paper.

You should make sure you answer all parts within each question so that the criterion can be assessed.

This examination is 3 hours in length. It is recommended that you spend approximately 35 minutes in total answering the questions in this booklet.

The 2013 External Examination Information Sheet for Computer Science can be used throughout the examination.

All written responses must be in English.

To be considered for a ‘C’ rating on a criterion, you must provide a satisfactory answer to at least the first question of the relevant section.

To be considered for a ‘B’ rating on a criterion, you must provide a satisfactory answer to at least the first two questions of the relevant section.

To be considered for an ‘A’ rating on a criterion, you must provide a satisfactory answer to all three questions of the relevant section.

You should show the methods used in deriving answers.

You should take care with the presentation of your answers, which should be complete and to the point. Diagrams should be used where appropriate. Complete sentences should be used in questions involving explanations. You are reminded that poor handwriting, spelling and expression that make it difficult for the examiners to understand what you mean may lead to lower marks.

A spare answer sheet has been provided in the back of the answer booklet for you to use if required.

If you use a spare answer sheet, you MUST indicate you have done so in your answer to that question.
Question 1

The following is a partially completed algorithm for an applet designed to calculate the cost of hiring a car. Users will enter the number of days, size of car and whether they want a GPS device (‘Y’ for yes) and the hire cost can then be calculated.

The numbers on the left of the algorithm are provided for reference purposes.

1  Initially
2    days = 0
3    size = 'S'
4    gps = 'Y'
5    cost = 0

6  When text is entered into “size” TextField
7    set size to text in “size” TextField

8  When a number is entered into “days” TextField
9    set days to value in “days” TextField

10 When text is entered into “GPS” TextField
11    set gps to text in “GPS” TextField

12 When the “Calculate” button is pressed
13    if size is ‘S’
14       set rate to 55
15    if size is ‘L’
16       set rate to 90
17    set cost to (rate x days)
18    if gps is ‘Y’
19       set gps_amount to zero
20    set cost to (cost + gps_amount )
21    display “Cost is” cost

The following intended features of the program do not work. Indicate the necessary changes to the algorithm so that these features work.

(a) A car can be hired for a maximum of 120 days. If the users enters more than 120 days then days is to be set to 120.

8  When a number is entered into “days” TextField
9    set days to value in “days” TextField
Question 1 (continued)

(b) The calculation of the hire cost needs to include the following:

(i) A rate of $70 per day should be included for a medium sized (‘M’) car.

(ii) If a user indicates a GPS device is required, a charge of $11 per day is applied.

12 When the “Calculate” button is pressed
**Question 2**

The following is a *partially completed* algorithm for an applet to be used to keep track of the player’s moves in a computer-based board game. The board is made up of 51 squares numbered 0 to 50. The player has two pieces that start on square 0.

The player is given two random numbers, both between 1 and 6. The player can then move the pieces forward by the number of squares indicated by the random numbers. Each random number can only be used once but can be used on either piece.

Once the two moves have been made the process is repeated until both pieces reach (or pass) square 50.

The number of turns taken is counted.

The numbers on the left of the algorithm are provided for reference purposes.

```
Initially
random1 = 0
random2 = 0
piece1 = 0
piece2 = 0
count = 0
move = 0
used1 = true
used2 = true
random_pick = false

When “Next random numbers” button is pressed
set random1 and random2 to random values between 1 and 6
display “Random numbers are ” random1, random2
set count to (count +1)
used1 = false
used2 = false

When “Random number 1” button is pressed
if used1 is false
    set move to random1
set used1 to true
set random_pick to true

When “Random number 2” button is pressed
if used2 is false
    set move to random2
set used2 to true
set random_pick to true
```

Question 2 continues opposite.
Question 2 (continued)

27 When “Piece 1” button is pressed
28     if random_pick is true
29         set random_pick to false
30         set piece1 to (piece1 + move)
31         display “Piece1 and Piece2 are ” piece1, piece2
32         if (piece1 >= 50) and (piece2 >= 50)
33             display “Task completed in ” count “ turns.”

34 When “Piece 2” button is pressed
35     if random_pick is true
36         set random_pick to false
37         set piece2 to (piece2 + move)
38         display “Piece1 and Piece2 are ” piece1, piece2
39         if (piece1 >= 50) and (piece2 >= 50)
40             display “Task completed in ” count “ turns.”

You are required to implement the following two additional rules:

(i) If one piece moves to the same square as the other piece, both pieces are returned to square zero.

(ii) If one piece moves past the other, it can advance one additional square.

The changes to be made to When “Piece 2” button is pressed are basically the same as those for When “Piece 1” button is pressed. You need only to show the changes to When “Piece 1” button is pressed.

27 When “Piece 1” button is pressed
Question 3

An applet is needed that calculates the cost of tickets to use the rides in an amusement park. The table below indicates the cost per ride for multi-ride tickets.

<table>
<thead>
<tr>
<th>Cost per ride*</th>
<th>1 to 10 rides</th>
<th>11 to 20 rides</th>
<th>more than 20 rides</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$5.00</td>
<td>$4.50</td>
<td>$4.00</td>
</tr>
</tbody>
</table>

* Rides are half-price if no scary rides are used.

The option of an unlimited rides ticket is also available at $100 (again this is half-price if no scary rides are used).

Children under 5 years of age have a 20% discount on all tickets but cannot use the scary rides.

Note:

- The applet should work correctly no matter what order the data is entered into the textfields.
- It can be assumed that all data entered is appropriate for each textfield.
- Use appropriate variable names.

In answering this question you must write the algorithm in part (a) and show the applet window design in part (b).

(a) Using the Initially/When model, write an algorithm for this applet.

.............................................................................................................................................
.............................................................................................................................................
.............................................................................................................................................
.............................................................................................................................................
.............................................................................................................................................
.............................................................................................................................................
.............................................................................................................................................
.............................................................................................................................................
.............................................................................................................................................
.............................................................................................................................................
.............................................................................................................................................
.............................................................................................................................................
.............................................................................................................................................

Question 3 continues opposite.
(b) Sketch the applet window for your algorithm, identifying all the text fields and/or buttons used.
This question paper and any materials associated with this examination (including answer booklets, cover sheets, rough note paper, or information sheets) remain the property of the Tasmanian Qualifications Authority.
Tasmanian Certificate of Education

COMPUTER SCIENCE

Senior Secondary

Subject Code: ITC315113

External Assessment

2013

Section B – Criterion 2

Time: approximately 35 minutes

On the basis of your performance in this examination, the examiners will provide results on the following criterion taken from the course statement:

Criterion 2  Demonstrate knowledge of a high level programming language.
CANDIDATE INSTRUCTIONS

Candidates MUST ensure that they have addressed the externally assessed criterion on this examination paper.

Answer ALL questions. Answers must be written in the spaces provided on the examination paper.

You should make sure you answer all parts within each question so that the criterion can be assessed.

This examination is 3 hours in length. It is recommended that you spend approximately 35 minutes in total answering the questions in this booklet.

The 2013 External Examination Information Sheet for Computer Science can be used throughout the examination.

All written responses must be in English.

To be considered for a ‘C’ rating on a criterion, you must provide a satisfactory answer to at least the first question of the relevant section.

To be considered for a ‘B’ rating on a criterion, you must provide a satisfactory answer to at least the first two questions of the relevant section.

To be considered for an ‘A’ rating on a criterion, you must provide a satisfactory answer to all three questions of the relevant section.

You should show the methods used in deriving answers.

You should take care with the presentation of your answers, which should be complete and to the point. Diagrams should be used where appropriate. Complete sentences should be used in questions involving explanations. You are reminded that poor handwriting, spelling and expression that make it difficult for the examiners to understand what you mean may lead to lower marks.

A spare answer sheet has been provided in the back of the answer booklet for you to use if required.

If you use a spare answer sheet, you MUST indicate you have done so in your answer to that question.
Question 4

(a) What will be the value of \( b \) and \( c \) after the following code is executed?

```java
double a, b, c;
a = 6;
b = a + 6 / 3;
c = Math.max(a, b);
```

Value of \( b \): ..........................................................................................................................................

Value of \( c \): ..........................................................................................................................................

Explanation:
...................................................................................................................................................
...................................................................................................................................................
...................................................................................................................................................

(b) What will be the value of \( q \) after the following code is executed?

```java
int q = 2;
int p = 7;
if (p > 7)
{
    p = p + 3;
}
switch (p)
{
    case 6: case 8: case 12:     q = 100;
        break;
    case 7: case 9: case 13:     q = 200;
        break;
    case 10: case 11:           q = 300;
        break;
}
```

Value of \( q \): ..........................................................................................................................................

Explanation:
...................................................................................................................................................
...................................................................................................................................................
...................................................................................................................................................
...................................................................................................................................................

Question 4 continues opposite.
Question 4 (continued)

(c) Trace the following code and find the final value of the variable \( n \).

```java
int n = 3;
boolean finished = false;
while (!finished) {
    n = n + 2;
    if (n > 6) {
        finished = true;
    }
}
```

Trace:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( finished )</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Final value of \( n \): ........................................................................................................
Question 5

(a) What will be the value of \( d \) and \( e \) after the following code is executed?

```java
int i;
double d, e;
d = 3;
i = 2;
d = 3 / i;
e = 3 / (double)i;
```

Value of \( d \): ........................................................................................................................

Value of \( e \): ........................................................................................................................

Explanation:
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................

(b) Trace the following code and find the final value of the variable \( s \).

```java
String s = "";
for (char k = 'A'; k < 'E'; k++)
{
    s = k + s;
}
```

Trace:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Final value of \( s \): ................................................................................................................
Question 5 (continued)

(c) Trace the following code and show what will be displayed in the applet window.

```java
int[] x = new int[] {50, 100, 150, 200, 250};
for (int j = 0; j < 4; j++)
    x[j] = x[j+1];
x[4] = 300;
for (int j = 0; j < 5; j++)
g.drawString(x[j] + "", x[j], 120);
```

Show your trace in the trace table below. Do not show the trace for the second for loop in the table.

<table>
<thead>
<tr>
<th>j</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

In the box below, draw the 400 × 200 applet window after the second for loop has been executed.
**Question 6**

This question relates to the following program. The numbers on the left are not part of the program and are provided for reference purposes. The program uses instructions in the array `instructions` to create a display in the applet window.

(a) In the method `move` there are two variables, `direction` and `d`. Both are used only in the method `move` but `d` is declared locally whereas `direction` is declared globally.

Explain why it is correct to declare `d` locally but not correct to declare `direction` locally.

(b) Explain how the contents of the instruction array will be interpreted by the `move` method to produce the final display in the applet window. Draw the pattern produced by the applet.

```java
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class Question6 extends Applet {
    int[] x, y;
    char direction;
    char[] instructions;
    boolean first;

    public void init()
    {
        x = new int[] {50, 50, 50, 50, 50, 50};
        y = new int[] {100, 120, 140, 160, 180, 200};
        instructions = new char[] {'L', 'F', 'L', 'R'};
        first = true;
    }
}
```

**Question 6 continues opposite.**
Question 6 (continued)

```java
public void move(int[] x, int[] y, char instruction)
{
    if (first)
    {
        direction = 'S';
        first = false;
    }
    char d = direction;
    switch (instruction)
    {
        case 'L' : switch (direction)
        {
            case 'N':  d = 'W'; break;
            case 'S':  d = 'E'; break;
            case 'E':  d = 'N'; break;
            case 'W':  d = 'S'; break;
        } break;
        case 'R' : switch (direction)
        {
            case 'N':  d = 'E'; break;
            case 'S':  d = 'W'; break;
            case 'E':  d = 'S'; break;
            case 'W':  d = 'N'; break;
        } break;
    }
    direction = d;

    for(int i = 0; i<=4; i++)
    {
        x[i] = x[i+1];
        y[i] = y[i+1];
    }

    switch(direction)
    {
        case 'N':  x[5] = x[4];
                    break;
        case 'S':  x[5] = x[4];
                    break;
                    y[5] = y[4];
                    break;
                    y[5] = y[4];
                    break;
    }
}

public void paint(Graphics g)
{
    for(int i = 0; i <= 3; i++)
        move(x, y, instructions[i]);

    for(int i = 0; i <= 5; i++)
        g.drawString("#", x[i], y[i]);
}
```
This question paper and any materials associated with this examination (including answer booklets, cover sheets, rough note paper, or information sheets) remain the property of the Tasmanian Qualifications Authority.
On the basis of your performance in this examination, the examiners will provide results on the following criterion taken from the course statement:

**Criterion 3** Use appropriate objects in the design of programs.
CANDIDATE INSTRUCTIONS

Candidates **MUST** ensure that they have addressed the externally assessed criterion on this examination paper.

Answer **ALL** questions. Answers must be written in the spaces provided on the examination paper.

You should make sure you answer all parts within each question so that the criterion can be assessed.

This examination is 3 hours in length. It is recommended that you spend approximately 35 minutes in total answering the questions in this booklet.

The 2013 External Examination Information Sheet for Computer Science can be used throughout the examination.

**All written responses must be in English.**

To be considered for a ‘C’ rating on a criterion, you must provide a satisfactory answer to at least the first question of the relevant section.

To be considered for a ‘B’ rating on a criterion, you must provide a satisfactory answer to at least the first two questions of the relevant section.

To be considered for an ‘A’ rating on a criterion, you must provide a satisfactory answer to all three questions of the relevant section.

You should show the methods used in deriving answers.

You should take care with the presentation of your answers, which should be complete and to the point. Diagrams should be used where appropriate. Complete sentences should be used in questions involving explanations. You are reminded that poor handwriting, spelling and expression that make it difficult for the examiners to understand what you mean may lead to lower marks.

*A spare answer sheet has been provided in the back of the answer booklet for you to use if required.*

*If you use a spare answer sheet, you MUST indicate you have done so in your answer to that question.*
Question 7

The following diagram shows an applet window with two buttons and three text fields (aButton, bButton, aField, bField and resultField). The ActionListener has been added to aButton and bButton.

The actionPerformed method for the applet is as follows:

```
public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == aButton)
    {
        int value = Integer.parseInt(aField.getText());
        value = value + 1;
        aField.setText(value + "");
        if (value == 3)
            resultField.setText("Won!");
    }
    if (e.getSource() == bButton)
    {
        int value = Integer.parseInt(bField.getText());
        value = value + 1;
        bField.setText(value + "");
        if (value == 2)
            resultField.setText("Lost!");
    }
}
```

When the text fields start with the values shown, what will be displayed in the text fields aField, bField and resultField after each of the following actions is executed?

(a) The initial values are:

```
| aButton | 0 | bButton | 0 | none |
```

The user clicks on aButton.

aField: ...............................................................................................................................

bField: ...............................................................................................................................

resultField: ...............................................................................................................................

Question 7 continues opposite.
Question 7 (continued)

(b) The initial values are:

The user clicks on \textit{bButton}.

\begin{itemize}
  \item aField: ........................................................ ...........................................................
  \item bField: ......................................................................................................................
  \item resultField: ...........................................................................................................
\end{itemize}

(c) The initial values are:

The user clicks on \textit{bButton}.

\begin{itemize}
  \item aField: ........................................................ ...........................................................
  \item bField: ......................................................................................................................
  \item resultField: ...........................................................................................................
\end{itemize}
Question 8

(a) Show the value of the variable `string3` each time it changes during the following section of code.

```java
String string1 = "This situation requires a ...";
String string2 = "Quest for Trolls!";
String string3;
string3 = string1.substring(0,5) + string2.substring(0,5);
int p = string1.indexOf("i",7);
string3 = string3 + string1.substring(p, p + 4);
p = string2.length();
string3 = string3 + string2.substring(10,p).toUpperCase();
```

First value of `string3`:

..........................................................................................................................................

Second value of `string3`:

..........................................................................................................................................

Third value of `string3`:

..........................................................................................................................................

(b) The following is a description of a Circle object:

<table>
<thead>
<tr>
<th>Field Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>int x</strong></td>
</tr>
<tr>
<td><strong>int y</strong></td>
</tr>
<tr>
<td><strong>int d</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Circle()</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setPosition(int x, int y)</td>
</tr>
<tr>
<td>void move(int x, int y)</td>
</tr>
<tr>
<td>void setDiameter(int d)</td>
</tr>
<tr>
<td>void enlarge(Circle c)</td>
</tr>
</tbody>
</table>
Question 8 (b) (continued)

The following code uses the Circle object and the drawOval method described at the bottom of this page.

```java
Circle c1 = new Circle();
Circle c2 = new Circle();
Circle c3 = new Circle();

c1.setPosition(400, 200);
c1.setDiameter(100);
g.drawOval(c1.x, c1.y, c1.d, c1.d);

c2.enlarge(c1);
c2.move(0, -50);
g.drawOval(c2.x, c2.y, c2.d, c2.d);

c3.enlarge(c1);
c3.enlarge(c3);
c3.move(-300, -150);
g.drawOval(c3.x, c3.y, c3.d, c3.d);
```

In the box below, draw the 800 × 600 applet window after the code is executed.

**View of applet window:**

Note: drawOval is a Graphics method that draws the outline of an oval as described below.

```java
public void drawOval(int x, int y, int width, int height)
```

**Parameters:**
- \( x \) – the \( x \) coordinate of the upper left corner of the oval
- \( y \) – the \( y \) coordinate of the upper left corner of the oval
- \( width \) – the width of the oval
- \( height \) – the height of the oval
Question 9

The class definition on page 9 defines a cataloguing system. It allows a number of categories to be set up and then each item can be placed in one of these categories.

(a) Using the class definition, write code to declare, instantiate and initialize a variable of the object type defined by the class.

..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................

(b) Using the variable from part (a), show how the methods in the class can be used to set up the two categories “Building” and “Office”.

Then add items “Plumber” and “Carpenter” to the “Building” category and “Clerk” to the “Office” category.

..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................

(c) Write code that uses the findCategory method to find the category to which the item “Clerk” belongs.

..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................
..........................................................................................................................................

Question 9 continues opposite.
Question 9 (continued)

(d) Write code for a `checkCategory` method within the class that will return `true` if the category specified by the parameter exists and `false` if it does not.

```java
public class Catalogue
{
    public String[][] store = new String[100][100];
    public Catalogue()
    {
        for (int i = 0; i<100; i++)
            for (int j = 0; j<100; j++)
                store[i][j] = "";
    }
    public void addCategory(String newCategory)
    {
        int i = 0;
        while (!store[i][0].equals(""))
            i=i+1;
        store[i][0] = newCategory;
    }
    public void addItem(String Category, String newItem)
    {
        int i = 0;
        while (!store[i][0].equals(Category))
            i=i+1;
        int j = 1;
        while (!store[i][j].equals("" ) )
            j=j+1;
        store[i][j] = newItem;
    }
    public String findCategory(String item)
    {
        boolean found = false;
        int i = 0;
        int j = 0;
        while (!found && !store[i][0].equals(""))
        {
            j = 1;
            while (!store[i][j].equals(item) && !store[i][j].equals(""))
                j=j+1;
            found = store[i][j].equals(item);
            i=i+1;
        }
        if (found)
            return store[i-1][0];
        else
            return "unknown";
    }
}
```
Tasmanian Certificate of Education

COMPUTER SCIENCE

Senior Secondary

Subject Code: ITC315113

External Assessment

2013

Section D – Criterion 4

Time: approximately 35 minutes

On the basis of your performance in this examination, the examiners will provide results on the following criterion taken from the course statement:

Criterion 4 Demonstrate knowledge and understanding of computer architecture.
CANDIDATE INSTRUCTIONS

Candidates **MUST** ensure that they have addressed the externally assessed criterion on this examination paper.

Answer **ALL** questions. Answers must be written in the spaces provided on the examination paper.

You should make sure you answer all parts within each question so that the criterion can be assessed.

This examination is 3 hours in length. It is recommended that you spend approximately 35 minutes in total answering the questions in this booklet.

The 2013 External Examination Information Sheet for Computer Science can be used throughout the examination.

**All written responses must be in English.**

To be considered for a ‘C’ rating on a criterion, you must provide a satisfactory answer to at least the first question of the relevant section.

To be considered for a ‘B’ rating on a criterion, you must provide a satisfactory answer to at least the first two questions of the relevant section.

To be considered for an ‘A’ rating on a criterion, you must provide a satisfactory answer to all three questions of the relevant section.

You should show the methods used in deriving answers.

You should take care with the presentation of your answers, which should be complete and to the point. Diagrams should be used where appropriate. Complete sentences should be used in questions involving explanations. You are reminded that poor handwriting, spelling and expression that make it difficult for the examiners to understand what you mean may lead to lower marks.

A spare answer sheet has been provided in the back of the answer booklet for you to use if required.

If you use a spare answer sheet, you **MUST** indicate you have done so in your answer to that question.
Question 10

(a) Complete the truth table for the expression:

\[ F \equiv (\neg B \land C) \lor (A \lor B) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>\neg B</th>
<th>\neg B \land C</th>
<th>A \lor B</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) Draw the logic circuit for the expression for G:

\[ G \equiv (\neg A \lor (B \land \neg A)) \land (C \lor B) \]

(c) Indicating the Logic Laws used, detail the steps which can be used to show:

\[ \neg Q \lor (P \land Q) \equiv \neg Q \lor P \]

\[ \neg Q \lor (P \land Q) \equiv \neg Q \lor P \]

\[ \neg Q \lor (P \land Q) \equiv \neg Q \lor P \]

\[ \neg Q \lor (P \land Q) \equiv \neg Q \lor P \]

\[ \neg Q \lor (P \land Q) \equiv \neg Q \lor P \]
Question 11

(a) This is a truth table for a logic device that has inputs $A$, $B$, $C$ and $D$ and a single output $H$.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
<th>$H$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(i) Complete the Karnaugh Map for $H$, from the truth table above.

(ii) Use the Karnaugh Map to produce a simple expression for $H$ and then draw a logic circuit for this expression.

Question 11 continues over the page.
Question 11 (continued)

(b) The following table contains a TOY machine code segment that implements an if statement.

Write java code, which includes only one if statement and uses the variables \( a \) and \( b \), that is the equivalent of this machine code.

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Contents</th>
<th>Pseudocode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0003</td>
<td>data</td>
<td>Variable ( a )</td>
</tr>
<tr>
<td>01</td>
<td>0000</td>
<td>data</td>
<td>Variable ( b )</td>
</tr>
<tr>
<td>02</td>
<td>0002</td>
<td>data</td>
<td>Constant 2</td>
</tr>
<tr>
<td>03</td>
<td>0004</td>
<td>data</td>
<td>Constant 4</td>
</tr>
<tr>
<td>10</td>
<td>8A00</td>
<td>R[A] ← mem[00]</td>
<td>Set Register A to the contents of location 00</td>
</tr>
<tr>
<td>11</td>
<td>8202</td>
<td>R[2] ← mem[02]</td>
<td>Set Register 2 to the Constant 2</td>
</tr>
<tr>
<td>13</td>
<td>CC17</td>
<td>if R[C] == 0 pc ← 17</td>
<td>Branch to address 17 if Register ( C = 0 )</td>
</tr>
<tr>
<td>14</td>
<td>DC17</td>
<td>if R[C] &gt; 0 pc ← 17</td>
<td>Branch to address 17 if Register ( C &gt; 0 )</td>
</tr>
<tr>
<td>15</td>
<td>8403</td>
<td>R[4] ← mem[03]</td>
<td>Set Register 4 to the Constant 4</td>
</tr>
<tr>
<td>16</td>
<td>9401</td>
<td>mem[01] ← R[4]</td>
<td>Store Register 4 in location 01</td>
</tr>
<tr>
<td>17</td>
<td>0000</td>
<td>exit</td>
<td>End of program.</td>
</tr>
</tbody>
</table>
Question 12

The registers in the CPU and the locations in the memory of a computer are both built using flip-flops.

(a) Why is the access time for data in a register significantly shorter than the access time for a location in memory?

............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................

In the TOY machine code the add instruction adds one number to another. Both numbers must be in registers in the CPU. Other machine code languages have an add instruction that can specify locations in memory.

(b) Explain how the TOY machine code can be used to add a number stored in memory to another number stored in memory.

............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................

Question 12 continues over the page.
Question 12 (continued)

(c) Consider the add instruction for TOY and an add instruction that can specify memory locations when adding two values from memory.

Explain, based on your understanding of the fetch/decode/execute cycle, why it could be faster to use the add instruction that specifies the memory locations.

............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
Spare Answer Sheet if Required

Question Number: 

(In the box write the number of the question you have answered.)

.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................
.......................................................................................................................................................

For Marker Use Only
This question paper and any materials associated with this examination (including answer booklets, cover sheets, rough note paper, or information sheets) remain the property of the Tasmanian Qualifications Authority.
On the basis of your performance in this examination, the examiners will provide results on the following criterion taken from the course statement:

**Criterion 5** Demonstrate knowledge and understanding of data representation and storage.
CANDIDATE INSTRUCTIONS

Candidates MUST ensure that they have addressed the externally assessed criterion on this examination paper.

Answer ALL questions. Answers must be written in the spaces provided on the examination paper.

You should make sure you answer all parts within each question so that the criterion can be assessed.

This examination is 3 hours in length. It is recommended that you spend approximately 35 minutes in total answering the questions in this booklet.

The 2013 External Examination Information Sheet for Computer Science can be used throughout the examination.

All written responses must be in English.

To be considered for a ‘C’ rating on a criterion, you must provide a satisfactory answer to at least the first question of the relevant section.

To be considered for a ‘B’ rating on a criterion, you must provide a satisfactory answer to at least the first two questions of the relevant section.

To be considered for an ‘A’ rating on a criterion, you must provide a satisfactory answer to all three questions of the relevant section.

You should show the methods used in deriving answers.

You should take care with the presentation of your answers, which should be complete and to the point. Diagrams should be used where appropriate. Complete sentences should be used in questions involving explanations. You are reminded that poor handwriting, spelling and expression that make it difficult for the examiners to understand what you mean may lead to lower marks.

A spare answer sheet has been provided in the back of the answer booklet for you to use if required.

If you use a spare answer sheet, you MUST indicate you have done so in your answer to that question.
Question 13

(a) Fill in the four missing bits in the following binary addition.

\[
\begin{array}{c}
1 & \square & 1 & 0 & 1 \\
+ & 1 & 1 & 1 & \square & 1 \\
\hline
1 & \square & 0 & \square & 1 & 0 \\
\end{array}
\]

(b) (i) Convert binary 101101 to hexadecimal.

......................................................................................................................................
......................................................................................................................................

(ii) If 101101 is stored in a 7-bit word using twos complement representation, what value does the 7-bit word represent?

......................................................................................................................................
......................................................................................................................................

(iii) If 01011101 is the code for an ASCII character, what is the character?

......................................................................................................................................
......................................................................................................................................

(c) Using the following table, convert 0.72 to binary with 6 places.

<table>
<thead>
<tr>
<th>Multiplication</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.72 x 2</td>
<td>1.44</td>
</tr>
<tr>
<td>0.44 x 2</td>
<td>0.88</td>
</tr>
<tr>
<td>0.88 x 2</td>
<td>1.76</td>
</tr>
<tr>
<td>0.76 x 2</td>
<td>1.52</td>
</tr>
<tr>
<td>0.52 x 2</td>
<td>1.04</td>
</tr>
<tr>
<td>0.04 x 2</td>
<td>0.08</td>
</tr>
<tr>
<td>0.08 x 2</td>
<td>0.16</td>
</tr>
<tr>
<td>0.16 x 2</td>
<td>0.32</td>
</tr>
</tbody>
</table>

......................................................................................................................................
......................................................................................................................................
......................................................................................................................................
Question 14

(a) The largest value that can be stored using a 16-bit word and twos complement representation is 32767. Explain how a floating point representation with the same number of bits can represent numbers much larger than 32767. No calculations are required in the answer.

(b) An 8-bit twos complement representation has a maximum value represented by 01111111. Give an examples of where the addition of two positive numbers could result in the representation of a negative number due to integer overflow.

(c) What would be the smallest number of bits necessary to represent the 26 letters of the alphabet in both upper and lower-case form? Explain your answer.
Question 15

(a) A TIFF file can store an image using the RGB [Red/Green/Blue] format or the CMYK [Cyan/Magenta/Yellow/Key (black)] format. This specifies the colour codes used to define the colour of a pixel in a TIFF image file. The Red, Blue and Green codes range from 0 to 255 whereas the Cyan, Magenta, Yellow and Key codes range from 0 to 100.

Determine the number of bits required to represent the 800 by 600 pixels in a TIFF image file using:

(i) the RGB format with the minimum number of bits used to store the values.

(ii) the CMYK format with the minimum number of bits used to store the values.

(ii) the CMYK format with a byte used to store each value.

(b) Why would the designers of the TIFF format be more likely to use bytes to store the values than to save space by using the minimum number of bits?

(c) An array is declared and instantiated as follows:

\[
\text{int[][] a = new int[2][5];}
\]

(i) Draw a diagram representing the array \( a \) that shows the links between all its components.

(ii) On your diagram, indicate the effect of the following java statement:

\[
a[1][2] = 4;
\]
Tasmanian Certificate of Education

COMPUTER SCIENCE

Senior Secondary

Subject Code: ITC315113

2013

Information Booklet
CONTENTS

Java
Introduction ......................................................................................................................................................... 3
Basics .................................................................................................................................................................. 4
Fundamentals ...................................................................................................................................................... 5
Event driven programming ................................................................. 10
Creating user interfaces with the AWT ......................................................... 11
Methods ............................................................................................................................................................ 14
Objects and classes ................................................................................................................................. 16

Computer Fundamentals and Computer Limitations
Number Systems ............................................................................................................................................... 22
Arithmetic in the Binary System .................................................................................................................. 23
Representation of Unsigned Integers ........................................................................................................... 23
Representation of Signed Integers ................................................................................................................ 24
Representation of Floating Point Numbers (Used For Reals) ............................................................................... 25
Representation of Characters and Strings .................................................................................................... 26
Representation of Boolean ................................................................................................................................ 26
ASCII Table ...................................................................................................................................................... 27
Representation of Arrays ................................................................................................................................ 28
Representation of Images and Sounds ........................................................................................................... 29
Logic Circuits ................................................................................................................................................... 30
List of Logic Laws ........................................................................................................................................... 31
Karnaugh Maps ................................................................................................................................................. 32
Design of a logic Circuit ................................................................................................................................... 37

Computer Architecture
The Toy Machine ................................................................................................................................................. 38
Structure of the CPU and the Machine Cycle .............................................................................................. 41
Java

Introduction

Java is platform independent. When you compile a Pascal program, it is translated into machine code or processor instructions specific to the processor on the machine that is running it. To use the program on another machine, a compiler must be obtained for that system. The Java compiler generates bytecodes. These instructions are not specific to one processor. The program is then executed by running a bytecode interpreter which reads the bytecodes and executes the program. This interpreter is often called the Java virtual machine.

For Java applets, the bytecode interpreter is built into Java-enabled browsers. For more general applications, an interpreter needs to be installed.

Java is an object-oriented programming language. It includes a set of class libraries that provide basic data types, system input and output capabilities and other utility functions.

Introduction - Hello World Applet

```java
import java.awt.*;
import java.applet.*;

public class HelloWorldApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello World", 5, 25);
    }
}
```

The HTML with the applet embedded

```html
<HTML>
<HEAD>
<TITLE>Computer Science Sample 1</TITLE>
</HEAD>

<BODY>
<APPLET CODE="HelloWorldApplet.class" WIDTH=150 HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```

Major Applets Activities

Initialization - occurs when the applet is first loaded. Occurs only once.
Starting - A startup behaviour for the applet that may occur many times (eg if applet is stopped.)
Stopping - Occurs when the reader leaves the page containing the running applets.
Destroying - enables the applet to clean up after itself (i.e. free up system resources)
Painting - The way the applet draws on the screen.

Web page is accessed and <applet> tag read - init( )
Applet is completely loaded and ready to run - start( )
User leaves Web Page - stop( )
User exits browser - destroy( )
Applet Parameters

Values can be passed to applets by adding parameters to the HTML code.

```java
import java.awt.*;
import java.applet.*;

public class HelloWorldApplet extends Applet {
    public void init() {
        name = getParameter("name");
        name = "Hello " + name;
    }

    public void paint(Graphics g) {
        g.drawString(name, 5, 25);
    }
}
```

The HTML with the applet embedded

```html
<HTML>
<HEAD>
<TITLE>Computer Science Sample 1</TITLE>
</HEAD>

<BODY>
<APPLET CODE="HelloWorldApplet.class" WIDTH=150 HEIGHT=25>
<Param Name=name VALUE="Babbage">
</APPLET>
</BODY>
</HTML>
```

NB The format of the command `g.drawString` is:

```
g.drawString(string_expression, x_coordinate, y_coordinate)
```

Java Basics

Keywords

<table>
<thead>
<tr>
<th>abstract</th>
<th>const</th>
<th>finally</th>
<th>int</th>
<th>public</th>
<th>throw</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>continue</td>
<td>float</td>
<td>interface</td>
<td>return</td>
<td>throws</td>
</tr>
<tr>
<td>break</td>
<td>default</td>
<td>for</td>
<td>long</td>
<td>short</td>
<td>transient</td>
</tr>
<tr>
<td>byte</td>
<td>do</td>
<td>goto</td>
<td>native</td>
<td>static</td>
<td>true</td>
</tr>
<tr>
<td>byvalue</td>
<td>double</td>
<td>implements</td>
<td>new</td>
<td>super</td>
<td>try</td>
</tr>
<tr>
<td>case</td>
<td>else</td>
<td>if</td>
<td>null</td>
<td>switch</td>
<td>void</td>
</tr>
<tr>
<td>catch</td>
<td>extends</td>
<td>import</td>
<td>package</td>
<td>synchronized</td>
<td>volatile</td>
</tr>
<tr>
<td>char</td>
<td>false</td>
<td>instanceof</td>
<td>private</td>
<td>this</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>final</td>
<td></td>
<td>protected</td>
<td>threadsafe</td>
<td>while</td>
</tr>
</tbody>
</table>

Page 4
Standard Packages and Classes

java.lang
Contains many standard classes including Integer, Double and math classes. This package is automatically available to programs and hence does not need to be imported.

java.awt

java.applet
This package contains one class Applet. It has all the functionality needed by an applet window.

java.awt.event
is used for implementing MouseListener, KeyListener and other event-driven program features.

java.awt.transfer
is used for transferring data to and from the clipboard.

java.io
is used for file input and output.

java.net
is used for performing network operations, such as sockets.

java.util
includes utilities for GregorianCalendar, Random and Vector.

Java Fundamentals

Variables and Data Types
Variable names start with a letter, underscore or dollar sign. Variable types can be one of the eight primitive data types, the name of a class or interface or an array.

<table>
<thead>
<tr>
<th>Integer</th>
<th>Floating Point</th>
<th>Char</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte (8 bits)</td>
<td>float (32 bits)</td>
<td>char (Unicode)</td>
<td>boolean</td>
</tr>
<tr>
<td>short (16 bits)</td>
<td>double(64 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int (32 bits)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>long (64 bits)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constants
We could declare pi as a variable but we may also wish to stop it being accidentally changed. We accomplish this by including the word final.

final double pi=3.142;
Comments

The symbols /* and */ are used to surround comments or // can be used for a single line comment.

Expressions and Operators

+  addition ==  equal to
-  subtraction !=  not equal
*  multiplication <  less than
/  division  >  greater than
%  modulus (integer remainder)  <=  less than or equal to

&&  AND
||  OR
^  XOR
!  NOT

(NB Result of / depends on the type of the original data.)

Promoting and casting

When dealing with expressions which contain different data types, Java will promote to the larger type and return an answer with that type.

e.g. 2 * 2.4 = 4.8

If we want the answer to be in the smaller type we cast down the hierarchy explicitly

e.g. (int)(2 * 2.4) = 4

Assignments

x=2+3;
x++; x--; (incrementing by one and decrementing by one)
x+=y; x-=y; x*=y; x/=y; (note: x+=y is the same as x=x+y)

Operator Precedence (Not a complete table)

.  []  ( )
++  --  !  ~  instanceof
new (type) expression
*  /  %
+  -
<,  >,  <=,  >=
==,  !=
&&
||
? : (Shorthand for if .. then .. else)
=,  +=,  -=,  /=,  %=,  ^=

Example

import java.applet.*;
import java.awt.*;

public class calculationExampleApplet extends Applet { /* Sample Java applet involving calculations */
public void paint (Graphics g) {
    g.drawString("5+6="+(5+6), 10,10);
    g.drawString("94.0/23.0"+(94.0/23.0),10,10);
}
}
**Mathematical Methods**

The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions. Note that methods of the same name may be defined for different types (int, long, float, double).

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(int a)</td>
<td>Returns the absolute integer value of a</td>
</tr>
<tr>
<td>abs(long a)</td>
<td>Returns the absolute long value of a</td>
</tr>
<tr>
<td>abs(float a)</td>
<td>Returns the absolute float value of a</td>
</tr>
<tr>
<td>abs(double a)</td>
<td>Returns the absolute double value of a</td>
</tr>
<tr>
<td>cos(double)</td>
<td>Returns the trigonometric cosine of an angle</td>
</tr>
<tr>
<td>exp(double a)</td>
<td>Returns the exponential number $e(2.718...)$ raised to the power of a</td>
</tr>
<tr>
<td>log(double a)</td>
<td>Returns the natural logarithm (base $e$) of a</td>
</tr>
<tr>
<td>max(int a, int b)</td>
<td>Takes two int values, a and b, and returns the greater of the two</td>
</tr>
<tr>
<td>max(long a, long b)</td>
<td>Takes two long values, a and b, and returns the greater of the two</td>
</tr>
<tr>
<td>max(float a, float b)</td>
<td>Takes two float values, a and b, and returns the greater of the two</td>
</tr>
<tr>
<td>min(int a, int b)</td>
<td>Takes two integer values, a and b, and returns the smaller of the two</td>
</tr>
<tr>
<td>min(long a, long b)</td>
<td>Takes two long values, a and b, and returns the smaller of the two</td>
</tr>
<tr>
<td>min(float a, float b)</td>
<td>Takes two float values, a and b, and returns the smaller of the two</td>
</tr>
<tr>
<td>min(double a, double b)</td>
<td>Takes two double values, a and b, and returns the smaller of the two</td>
</tr>
<tr>
<td>pow(double a, double b)</td>
<td>Returns the number a raised to the power of b</td>
</tr>
<tr>
<td>random()</td>
<td>Generates a random number between 0.0 and 1.0</td>
</tr>
<tr>
<td></td>
<td>eg. (int)(Math.random() * 4 + 2) will give one of the values 2, 3, 4 or 5</td>
</tr>
<tr>
<td>round(float)</td>
<td>Rounds off a float value by first adding 0.5 to it and then returning the</td>
</tr>
<tr>
<td></td>
<td>largest integer that is less than or equal to this new value</td>
</tr>
<tr>
<td>round(double)</td>
<td>Rounds off a double value by first adding 0.5 to it and then returning the</td>
</tr>
<tr>
<td></td>
<td>largest integer that is less than or equal to this new value</td>
</tr>
<tr>
<td>sin(double)</td>
<td>Returns the trigonometric sine of an angle</td>
</tr>
<tr>
<td>sqrt(double)</td>
<td>Returns the square root of a</td>
</tr>
<tr>
<td>tan(double)</td>
<td>Returns the trigonometric tangent of an angle</td>
</tr>
<tr>
<td>toDegrees(double)</td>
<td>Translates radians to degrees</td>
</tr>
<tr>
<td>toRadians(double)</td>
<td>Translates degrees to radians</td>
</tr>
</tbody>
</table>

**Example**

```java
import java.awt.*;
import java.applet.*/

/*/ Sample Java applet involving mathematical functions *//

public class Math_functions extends Applet {

    public void init() {
    }

    public void paint(Graphics g) {
        g.drawString("The larger number of 10 and 20 is "+Math.max(10,20), 50, 50);
        g.drawString("A random number between 1 and 6 is "+((int)(Math.random() * 6)+1), 50, 80);
    }
}
**Block Statements**

Blocks in Java are enclosed between `{ }`

**If ... else Statement (Conditional)**

if (hours>40) overtime=hours-40;

if (hours>40)  
overtime=hours-40;
else overtime=0;

if (hours>40)  
{overtime=hours-40;  
 normal=40;}
else  
{overtime=0;  
 normal=hours;}

**Switch Conditional**

switch (oper) {
  case '+':  
g. drawString("5+6=","+(5+6), 10,10);  
  break;
  case '-':  
g. drawString("5-6=","-(5-6),10,10);  
  break;
  case '*':  
g. drawString("5*6=","*(5*6),10,10);  
  break;
  default : g.drawString("wrong operator", 10,10);  
  break;
}

**For Loops**

for (initialisation; test; increment) {
  statements;
}

Eg. for (int i=1; i<=100; i=i+10)  
g.drawString("Computing is great", i,10);

**While Loops**

while (condition) {
  statements;
}

**Do .... while loops**

do {
  statements;
}  
while (condition);

Note: Can only be used with integer and char datatypes
Note: use of break to exit the switch statement and send control immediately after }
Note: can have a default case which handles all other values e.g.
default : g.drawString("wrong operator", 10,10);
  break;

Note: tests before entering the loop each time

Note: test is performed before entering the loop - if the condition is false when the loop is first entered it will not be executed.

Note: test is performed after body of loop is executed so it is always performed at least once.
Arrays and Subscripts

You can create arrays of any type or object e.g. strings, buttons, images

int xcoordinate[] = {0,10,20,30,40,50,60,70,80,90};
or
int xcoordinate[] = new int[10];
xcoordinate[0]=0; xcoordinate[1]=10; etc

The array can be processed in loops

e.g.
for (int i=0; i<10; i++) {
g.drawString("\r", xcoordinate[i],100);
}

Two Dimensional Arrays

int scores[][] = new int[3][5];

int scores[][] = {{93,74,77,55,81},
                 {78,77,72,75,80},
                 {92,88,82,83,69}}

and processed within nested loops using variables of type scores [i][j]

Strings

(Note: single quotes are used around char type and double quotes around strings)

String subject="Computer Science";
String addon="is great";
Concatenation: use +  (eg subject="Computer" + " " + "science")
Comparison: don’t use ==
string1.equals(addon)
string1.equalsIgnoreCase(subject)
string1.compareTo(string2) results in..

• 0 if the strings are equal;
• a negative value if the string object preceded the parameter;
• a positive value if the string object follows the parameter.

Amending:
String2=string1.replace(‘e’,’a’)
String2=string1.toUpperCase()  string2=string1.toLowerCase()  
subject.trim()      //used to move trailing or leading blanks

Examining Strings:
subject.length() is 16
subject.substring(3,6);    //will return “put” - the characters from starting position 3 to 1 greater than the
                          last character to be extracted. (The first character position is 0.)
sObject.charAt(1);   //will give the character at position 1 (resulting in a char datatype)
sObject.indexOf("put",2)  //will return 3, i.e. the position of the first letter of the substring “put”, starting
                          looking at position 2 – will return -1 if the string is not found.
"c:/directory/file".lastIndexOf("/");  //results in 12 i.e. the last occurrence of the substring "/"
subject.endsWith("r")   //would result in a true boolean result because the string ends with “r”
subject.startsWith("X")  //would result in a false boolean result

Notice the subscripts begin at 0 i.e. the array goes from xcoordinate[0] to xcoordinate[9].
Converting Strings

to convert an int to a String:
int n = 123;
String s = Integer.toString(n);  // s becomes “123”

to convert a float to a String:
float f = 12.34f;
String s = Float.toString(f);  // s becomes “12.34”

to convert a String to an int:
String s = “1234”;
int n = Integer.parseInt(s);

to convert a String to a float:
String s = “12.34”;
Float temp = Float.valueOf(s);
float f = temp.floatValue();

Event Driven Programming

A key component of a GUI is event-driven programming. Therefore the program needs to recognise an event and respond. The program is said to be listening for events. This is done using

implements specific listener

The listener interfaces include:

**ActionListener**

Event  public void actionPerformed(ActionEvent e)  (any component event)

**MouseListener**

mouse down  public void mousePressed(MouseEvent e)  (left button down)
mouse up  public void mouseReleased(MouseEvent e)  (left button up)
mouse clicks  public void mouseClicked(MouseEvent e)  (left button down then up)
mouse enter  public void mouseEntered(MouseEvent e)  (mouse enters component area)
mouse exit  public void mouseExited(MouseEvent e)  (mouse exits component area)

Once the interface has been added using implements, stubs are entered for all of the events in the interface.

e.g. if using MouseListener interface we need

        public void mousePressed(MouseEvent e) { }
        public void mouseReleased(MouseEvent e) { }
        public void mouseClicked(MouseEvent e) { }
        public void mouseEntered(MouseEvent e) { }
        public void mouseExited(MouseEvent e) { }

Detail is then filled in the event which is of interest.

**ItemListener**

        public void itemStateChanged(ItemEvent e)
Finally, the listener must be registered.
e.g. addMouseListener( )

Example: a program to draw an oval where the user clicks on the screen.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class event1 extends Applet implements MouseListener {
    int x=30;
    int y=30;

    public void init() {
        setBackground(Color.blue);
        addMouseListener(this);
    }

    public void mouseReleased(MouseEvent e) {
    }

    public void mouseClicked(MouseEvent e) {
    }

    public void mouseEntered(MouseEvent e) {
    }

    public void mouseExited(MouseEvent e) {
    }

    public void paint(Graphics g) {
        g.fillOval(x,y,50,50);
    }

    public void mousePressed(MouseEvent e) {
        x=e.getX();
        y=e.getY();
        repaint();
    }
}

Note: These need to be present, even though they are not going to be used.

Creating User Interfaces with the AWT

The major components of the AWT are

Containers - components that contain other components
Canvases - simple drawing surface
UI components - buttons, lists, pop-up menus, check boxes, text fields etc.
Window Construction components - windows, frames, menu bars and dialog boxes.

Applets are automatically AWT containers and hence components can be added to them.

Labels

Labels can be created in the following ways.

Label() Constructs an empty label.
Label(String) Constructs a new label with the specified string of text, left justified.
Label(String, int) Constructs a new label that presents the specified string of text with the specified alignment.
Label prompt;

    public void init()
    {
        prompt= new Label("Please enter number");
        add(prompt);
    }

Label Methods

getText( ) Returns a string containing label text
setText(String) Changes text of label
getAlignment( ) Returns an integer representing alignment of the label
setAlignment(int) Changes label alignment

Buttons

public void init()
{
    add(new Button("Award=OA"));
}

setLabel(String label) Sets the button's label to be the specific
getLabel() Gets the label of this button.

Making component invisible

cOMPONENTNAME.setVisible(true/false) will be visible if true, not visible if false

Check Boxes

Used to allow a user to specify one or more choices by clicking on a box. Implements ItemListener, handles events with ItemStateChanged method.

for example:
Checkbox burger, drink;

public void init () {
    burger = new Checkbox("Burger", true); // sets original state of checkbox to true i.e., ticked
drink = new Checkbox("Drink");

    add(burger);
    add(drink);

    burger.addItemListener(this);
drink.addItemListener(this);
}

Check Box Methods

Checkbox( ) Creates an empty checkbox
Checkbox(String) Creates a checkbox with string as a label
Checkbox(String, boolean) Creates a checkbox that is either selected or deselected
getLabel( ) Returns the label as a string
setLabel( ) Changes text of checkbox
getState( ) Returns true or false based on whether checkbox is selected.
setState(boolean) Changes state of check box
Radio Buttons (called Checkbox Groups in AWT library)

To add radio Buttons - used when there are a limited number of mutually exclusive choices
e.g. credit card options - one of Visa, MasterCard or Bankcard
Implements ItemListener, handles events with ItemStateChanged method.

for example:
CheckboxGroup creditCardType= new CheckboxGroup( );

public void init() {
    visa=new Checkbox(“Visa”,creditCardType,false);
    add (visa);
    visa.addItemListener(this);

    …similarly for other credit card options)
}

Choice Menus

These are pop up or pull down menus. An item can be selected by clicking on it. The selected item appears
as the only visible part of the menu.
Implements ItemListener, handles events with ItemStateChanged method.

for example:
public void init( ) {
    Choice icecream=new Choice( );
    icecream.add("Chocolate");
    icecream.add("Vanilla");
    icecream.add(“Strawberry”);
    add(icecream);
    icecream.addItemListener(this);
}

Choice Menu Methods

getItem(int) returns string item at the given position
getItemCount( ) returns the number of items in the menu
getSelectedIndex( ) returns the index position of the selected item
getSelectedItem( ) returns the currently selected item as a string
select(int) selects the item at a given position
select(String) selects the item with the given string

Lists

These are a list of text strings from which one or more can be selected. A scrollbar is provided to scroll up
or down the list.
Implements ActionListener, handles events with actionPerformed method.
(This responds to double-clicks)

for example:
List colourList = new List(4,false);
colourList.add("red");
colourList.add("blue");
……………..
add(colourList);
colourList.addActionListener(this);

Two parameters: e.g. 4,false
- how many items are visible at once
- false→ only one item can be selected
  true→ more than one item can be selected
Scrollbar

Implements AdjustmentListener, handles events with adjustmentValueChanged method.

for example:
Scrollbar slider = new Scrollbar(Scrollbar.VERTICAL, initial, visible, min, max)
add(slider);
slider.addAdjustmentListener(this);

Text Fields

Allow you to enter and edit text.

TextField(int) creates text field of int characters wide
TextField(String, int) creates text field containing string of int characters wide
TextField(String) creates a text field containing string.

Text Field Methods

getText returns the text that the text field contains
setText(String) puts the given text into the string
getColumns( ) returns the width of the text field
select(int, int) selects the text between the two positions
selectAll( ) selects all text in field
isEditable( ) returns true or false
setEditable(boolean) true enables text field to be edited
getEchoChar(char) returns the character used for masking input
setEchoChar(char) sets the character used for masking input
echoCharIsSet( ) returns true or false

Other features include canvases and cursors.

Methods

Methods have a number of important uses:

- They enable you to break up a class into “modules” which each perform a distinct task. Classes written in this way are much easier to implement and to follow.
- They enable the code for a repetitive task to be written once, and used many times in a class.
- In an object oriented language, they provide the means of using the information that is stored in an object. They also enable you to use the features of one class in another (such as when you call Math.random()).

The standard Applet methods

Three methods are run automatically in order. If we don't further define these we get an empty version by default: in this case, they do not have to be written in your class code – they run automatically anyway.

- init: init is used to set up the GUI (Graphical User Interface) by instantiating and adding components, define the initial values of variables, tell actionPerformed() which events to listen to and set up sounds and images to be used by the Applet.
- start: We will never use this method in this course. However, it can be modified for more advanced purposes.
- paint: This is used to display graphics objects. It runs automatically after the start () method, and you sometimes have to be careful not to write out “junk” this first time around. You can run paint again
The general form of a method **definition**.

```
public   return_value_type    method_name    ( formal parameters )  {
  declarations;  //These are local variables or objects.
  statements;
  return <expression>;   //If required. Must be included if the method is not void.
}
```

**Scope Rules**

Variables can be declared anywhere however they are usually declared in one of three places as follows.

They can be declared just inside the class definition. These variables are known as **instance** or **global** variables and may be used within any method defined within the class. (**class scope**)

```
public class Methods extends Applet
(TextField t1;
```

Variables can be declared in the parameter list of a method definition. They are known as **parameters** or **local variables** and are used within the method. (**Block scope**)

```
public void stars(int number)
```

Finally variables can be declared within a method definition and are known as local variables. They may only be used within the method.

```
public String stars(int number, char letter)
{String temp;
```

When a local variable has the same name as an instance variable or a parent method, it is hidden until the child method terminates. The instance variable however can be accessed using this.instance variable name.

**Passing parameters**

**Pass-by-value vs pass-by-reference**

When arguments of a **basic data type** are passed to a method they are **pass-by-value**. This means that for each argument passed to a method, a copy is made and it is the copy that is passed and referenced through the parameter made, not the original value. Consequently, whatever the method does to the parameter, it has no effect on the value in the invoking method.

```
e.g.
int i=10;
int x=change(i);
```

This method will return 11 but the original value of i = 10 is unchanged.

```
public int change (int j)
{  j++;
  return j;
}
```

However, objects are different. When **objects** are passed to a method they are **pass-by-reference**. A reference is not a copy of the object, but a pointer to the object. So if and when a method changes an object parameter, it changes the object in the piece of the program that invoked the method.
Passing strings as parameters

Strings are objects and are therefore passed by reference. But, changing the value of a string is regarded as creating a totally new string, rather than manipulating an existing value via methods. The change therefore has only a local effect and we can think of strings as behaving like a basic data type being passed by value.

Passing arrays as parameters

An array is an object in Java. When an object is used as a parameter it is passed as a reference. **So when an array is passed as a parameter to a method, a reference to the array is passed** (i.e. a pointer to the array and not a copy of the array). So when a method changes an array parameter, it changes the array in the piece of program that invoked the method.

e.g.

```java
int []table = new int[8];
fillZero(table);
```

*with the method:*

```java
public void fillZero(int[]array) {
    for (int s=0; s<array.length; s++)
        array[s]=0; }
```

This method will actually change the values in the array `table` to zero (0).

Data modelling (Encapsulation) in an object-oriented programming (OOP).

Object-oriented programming is an attempt to model computer programs as closely as possible on objects in the real world. Each object has variables that keep track of what is going on inside the object and methods to allow other objects to communicate with it.

Related data and methods are grouped together in an object in such a way that a programmer can use the object without having to worry about its details, or running the risk of destroying it.

Data and methods which the user does not need to worry about are “hidden” (declared as “private”) while “public” data and methods are available in a way which are simple and safe. In other words, objects should not be able to change (or even look directly at) each other’s instance variables but should use the object’s methods. It allows pre-defined data models such as String to be used without having to know how the string of text is stored or manipulated.

**Objects and Classes**

Object

- A combination of data (variables to keep track of what is happening) and methods (a process to let objects communicate) used on that data.
- Objects that we have dealt with include strings, the various applet components such as buttons, colors, fonts and so on.
- As an example if stringVar is an object of the String class, the data stored in stringVar is the text such as “Hello” and the methods are all the methods associated with the String class, such as stringVar.length();

Class

A general computer program which carries a “design” or “blueprint” or “template” for an object, and so enables a programmer to create or “instantiate” any number of objects of this class.
Instantiating an object

When you want to use an object in a class you must:

- **Declare** the object by giving it’s class and identifier name:
  
  eg Button myButton;

  By convention the class name begins with a capital while the object name begins lower case.

- **Instantiate** the object by calling one of the constructor methods for the object with the `new` method.
  
  Classes will often have a number of different constructors which take different parameters, and you can look up the constructors for different objects in the Java API help. The constructor method always has the same name as the class, but there may be any number of parameters. Examples are:
  
  myButton = new Button(“Hello”);

Creating a class

The class must have the following structure:

```java
public class <name of Class>
{
    <declarations of variables to be used>
    <constructor method>
    <method declarations>
}
```

- A variable or method in a class can have one of 3 types of access:
  
  ```java
  public accessible from any class.
  private accessible only from within this class.
  (protected not used in this course.)
  ```

  Remember that local variables are never accessible from outside a particular method.

- The prefix `final` means that a variable or method cannot be changed (variable) or overwritten (method).

- The **instance variables** declared in a class block are generally defined as private and can’t be used outside the class. They give a way of keeping track of the state of an object.

  ```java
eg class Car
    {
        int   maxSpeed;
        int   weight;
        String  make;
    }
```

  Every car created within the program must have a maxSpeed, weight and make.

- The **constructor method** is a very special method used to create (instantiate) and initialise an object and it does not return a value. It has the same name as the class. It is recommended to set the instance variables to default values.

  ```java
  eg public Car()
    {
        int   maxSpeed = 0;
        int   weight = 0;
        String  make= null;
    }
```

  If you want to specify initial values when an instance of an object is made, you must use parameters.

  ```java
  eg public Car(int maxSpeed, int weight, String make)
    {
        maxSpeed = 0;
        weight = 0;
        make= null;
    }
```

  So in init() this constructor would be used  
  
  fredsCar = new Car(500, 1000, “Honda”);

- The **method declarations** are the methods that can be used on Objects from this class. They are blocks of code that can be called from outside the object and can take arguments and optionally return a value.
The method modifiers allow you to say how restrictive the method will be. If it is public, any class can access the method.

Continuing with the car theme

```java
Car fredsCar;    // declare the object
fredsCar = new Car();   // instantiate fredsCar
fredsCar.getSpeed();   // accessing one of the car methods by using the object
// followed by the dot operator

// An Example of a complete Car program

public class Car
{
    int maxSpeed;
    int weight;
    String make;
    public Car()    // car constructor
    {
        maxSpeed = 0;
        weight = 0;
    }
    public void setMaxSpeed(int speed)
    {
        maxSpeed = speed;
    }
    public int getSpeed()
    {
        return maxSpeed;
    }
    public void setWeight(int heavy)
    {
        weight = heavy;
    }
    public int getWeight()
    {
        return weight;
    }
    public void setMake(String carMake)
    {
        make = carMake;
    }
    public String getMake()
    {
        return make;
    }
}    // end Car
```

public class CarSample extends Applet
```java
{  
  Car fredsCar;
  public void init()
  {
    setBackground(Color.white);
    fredsCar = new Car();
    fredsCar.maxSpeed(500);
    fredsCar.setweight(1000);
    fredsCar.setMake("Honda");
  }
  public void paint(Graphics g)
  {
    int speed;
    int weight;
    String make;
    speed = fredsCar.getSpeed();
    weight = fredsCar.getWeight();
    make = fredsCar.getMake();
    g.drawString("Fred's car Speed " + speed + "weight" + weight + "make" + make, 10, 20);
  }
} // end CarSample

A second example with 2 constructors for the Car

public class Car
{
  int maxSpeed;
  int weight;
  String make;
  public Car() // car constructor 1
  {
    maxSpeed = 0;
    weight = 0;
  }
  public Car(int speed, int carWeight, String carMake) // car constructor 2
  {
    maxSpeed = speed;
    weight = carWeight;
    make = carMake;
  }
  public void setMaxSpeed(int speed)
  {
    maxSpeed = speed;
  }
  public int getSpeed()
  {
    return maxSpeed;
  }
  public void setWeight(int heavy)
  {
    weight = heavy;
  }
  public int getWeight()
  {
    return weight;
  }
  public void setMake(String carMake)
```
```java
{    make = carMake;
}
public String getMake()
{    return make;
}
} // end Car

public class CarSample extends Applet
{
    Car  fredsCar, marysCar;
    public void init()
    {
        setBackground(Color.white;
        fredsCar = new Car( );
        fredsCar.maxSpeed(500);
        fredsCar.setweight(1000);
        fredsCar.setMake("Honda");
        marysCar = new Car(600, 900, "Porsche"); // using constructor 2
    }
    public void paint(Graphics g)
    {
        int  speed;
        int  weight;
        String  make;
        speed = fredsCar.getSpeed();
        weight = fredsCar.getWeight();
        make = fredsCar.getMake();
        g.drawString("Fred’s car speed “ + speed + “ weight ” + weight + “ and make ” + make, 10, 20);
        g.drawString("Mary’s car speed “ + marysCar.getSpeed + “weight ” + marysCar.getWeight + " and make ” + marysCar.getMake, 10, 40);
    }
} // end CarSample

Creating your own Data Model

To create your own Data Model you create class and objects from the class. In doing so you need to consider what data and methods you will require. For example to create a class to represent a die we may need:

Data such as
  face
    the current value of the die.

Methods such as
  Die()
    Constructor method to create a new dice object with an initial random value
  throwDie()
    Imitates throwing a die and generates a new value for the die
  dieValue()
    Returns the current value of the die
  isEven()
    Returns a boolean value indicating if the die value is an even number.
```
The following program `CircleObject` uses a data model for a circle by creating a Circle class so Circle Objects can be created in the program and you can change their size and display them on the screen.

```java
import java.awt.*;
import java.applet. *

public class CircleObject extends Applet {
    Circle C1, C2;

    public void init() {
        C1 = new Circle(100,100,50);
        C2 = new Circle(200,200,30);
    }

    public void paint(Graphics g) {
        g.drawString("Showing circle outputs", 20, 20);
        C1.drawCircle(g);
        C2.drawCircle(g);
        C1.changeSize(25);
        C1.drawCircle(g);
    }

    class Circle {
        private int radius;
        private int xCoord, yCoord;
        public Circle(int initialX, int initialY, int initialRadius) { //constructor
            radius = initialRadius;
            xCoord = initialX;
            yCoord = initialY;

            public void changeSize(int size) {
                radius = radius + size;
            }
            public void drawCircle(Graphics g) {
                g.drawOval(xCoord, yCoord, radius*2, radius*2);
            }
        }
    }
}
```
Computer Fundamentals and Computer Limitations

Number Systems

A numbering system is a way of representing numbers. The decimal (base 10) system for counting was used because we have 10 digits (fingers). Computers use the binary (base 2) system because it is easy to produce electronic components which are in one of 2 states (on / off) rather than 10 states. Hexadecimal (base 16) is a simple compact way to represent a binary number.

Converting from binary to decimal

The easiest way to convert from binary to decimal (with small numbers) is to refer to the table below:

<table>
<thead>
<tr>
<th>2^9</th>
<th>2^8</th>
<th>2^7</th>
<th>2^6</th>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
<th>2^-1</th>
<th>2^-2</th>
<th>2^-3</th>
<th>2^-4</th>
<th>2^-5</th>
<th>2^-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>256</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>.5</td>
<td>.25</td>
<td>.125</td>
<td>.0625</td>
<td>.03125</td>
<td>.015625</td>
</tr>
</tbody>
</table>

eg 1012 has the value $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1 = 5$

11002 has the value $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 4 = 12$

0.1012 has the value $1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = .5 + .125 = 0.625$

Converting from decimal to binary

Convert 21110 to binary.

<table>
<thead>
<tr>
<th>Dividends</th>
<th>Remainders</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2111</td>
</tr>
<tr>
<td>2</td>
<td>105</td>
</tr>
<tr>
<td>2</td>
<td>52</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Hence $211_{10} = 11010011_2$

Converting the fractional parts from decimal to binary

eg $0.25_{10}$ to binary.

$0.25 \times 2 = 0 + 0.5$  \(\text{first digit} = 0\)

$0.5 \times 2 = 1 + 0$  \(\text{second digit} = 1\)

Giving $0.25_{10} = 0.01_2$

eg $0.1_{10}$ to binary.

$0.1 \times 2 = 0 + 0.2$  \(\text{first digit} = 0\)

$0.2 \times 2 = 0 + 0.4$  \(\text{second digit} = 0\)

$0.4 \times 2 = 0 + 0.8$  \(\text{third digit} = 0\)

$0.8 \times 2 = 1 + 0.6$  \(\text{fourth digit} = 1\)

$0.6 \times 2 = 1 + 0.2$  \(\text{fifth digit} = 1\)

$0.2 \times 2 = 0 + 0.4$  \(\text{sixth digit} = 0\)

etc etc.

Writing as a recurring binary number:

$0.1_{10} = 0.0001100110011\ldots_2$

Notes

Successive division continues until the quotient becomes zero. The number to the new base is then determined by taking the remainders from last to first.

Notes

An exact decimal fraction may not have an exact binary equivalent. For divisions which do not go exactly go to one more place than asked for, then round.

The decimal number 0.1 cannot be represented exactly in binary. It must be rounded off to a finite number of decimal places, giving rise to round-off error.
Rounding of to, say, 6 binary points:

\[ 0.1_{10} = 0.000110_2 \]

**eg 3** 5.125 to binary

- 5 becomes 101 in binary
- .125 × 2 = 0 + .25 \text{ first digit} = 0
- .25 × 2 = 0 + .5 \text{ second digit} = 0
- .5 × 2 = 1 + 0 \text{ third digit} = 1

Thus 5.125\(_{10}\) = 101.001\(_2\)

### Converting from binary to hexadecimal

Because 16 is equal to \(2^4\) each hexadecimal digit represents 4 binary digits. Thus to convert a binary number to hexadecimal the number is divided into groups of four binary digits. Each of these groups is then converted to its hexadecimal equivalent from the table on the right.

For example the binary number

\[ 01001111010101_2 \]

is divided as

\[ 0100 1111 0101_2 \]

and these groups are then converted to Hexadecimal

\[ 13D\_5_{16} \]

### Arithmetic in the Binary System

#### Addition

Arithmetic in binary follows exactly the same rules as for decimal arithmetic.

**eg**

\[
\begin{array}{c}
\text{Decimal} \\
\text{Binary} \\
\text{Hexadecimal}
\end{array}
\begin{array}{c|c|c}
0 & 0000 & 0 \\
1 & 0001 & 1 \\
2 & 0010 & 2 \\
3 & 0011 & 3 \\
4 & 0100 & 4 \\
5 & 0101 & 5 \\
6 & 0110 & 6 \\
7 & 0111 & 7 \\
8 & 1000 & 8 \\
9 & 1001 & 9 \\
10 & 1010 & A \\
11 & 1011 & B \\
12 & 1100 & C \\
13 & 1101 & D \\
14 & 1110 & E \\
15 & 1111 & F \\
\end{array}
\]

- \(1 + 0 = 1\)
- \(0 + 1 = 1\)
- \(1 + 1 = 0 + \text{carry of 1}\)
- \(1 + 1 + 1 = 1 + \text{carry of 1}\)

### Representation of Unsigned Integers

It wastes storage to represent numbers character by character, because **one byte** is used for **each** digit. So numbers are converted to binary and represented in that form.

One **8-bit byte** can represent numbers ranging from \(0000\ 0000_2 = 0_{10}\) to \(1111\ 1111_2 = 255_{10}\)

This range is not generally large enough to show

- integers greater than 255
- negative numbers

The group of bits that the computer handles as a whole is referred to as a **word** (ie a 16-bit machine can handle 16 bits at a time).

\[ \begin{array}{c|c|c}
15 & 1-14 & 0 \\
\end{array} \]
Bit 15  Most significant bit  eg a 16-bit word
Bits 1-14  Balance of the word
Bit 0  Least significant bit

The largest integer in a 16-bit word is $2^{16} - 1$ (65535).

To show larger integers you need a longer word but there is always some fixed limit to integer size and the need for a larger integer would cause the system to break down.

If it is not possible to store the results of an operation the system 'overflows' the register provided, this is arithmetic overflow.

### Representation of Signed Integers

#### Twos Complement

The twos complement of a binary value is the complement of the value with respect to $2^n$ where $n$ is the number of bits in the word.

eg. Consider 1010, now $n = 4$ therefore $2^4 = 10000$ hence complement is $10000 - 1010 = 0110$

**A simpler theory:**

A method to obtain twos complement

1. Take the one’s complement and add “1”.
2. A shortcut technique works as follows:
   - Start from the right of the number and work towards the left.
   - Any ‘0’ bits remain the same until the first ‘1’ bit and the first ‘1’ bit remains the same.
   - After the first ‘1’ bit all subsequent bits are reversed.

Examples (assuming an 8 bit twos complement representation):

<table>
<thead>
<tr>
<th>Original</th>
<th>Binary representation</th>
<th>2’s complement</th>
<th>New value</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0000 0110</td>
<td>1111 1010</td>
<td>-6</td>
</tr>
<tr>
<td>-6</td>
<td>1111 1010</td>
<td>0000 0110</td>
<td>6</td>
</tr>
</tbody>
</table>

#### Interpreting numbers in twos complement representation

**Positive integers**
- The most significant bit is “0”.
- The value is equal to the binary string. For example, 0000 0011 represents $3_{10}$.

**Negative integers**
- The most significant bit is “1”.
- To obtain the value, “take a twos complement”. For example 11110110 represents a negative value. Taking a twos complement gives 00001010 which represents the positive number $6_{10}$. Hence 11110110 represents $-6_{10}$.

#### Arithmetic in twos complement representation

Addition using the twos complement representation the same as addition of binary numbers except that the size of the number is limited by the number of bits in the representation. If the sum is too big it will cause an overflow error.

Subtraction is done by adding a negative value. That is to carry out the operation $7 - 5$ the computer will calculate $7 + (-5)$. The (-5) is represented by the twos complement of 5.
For example using an 8 bit word:

\[
\begin{array}{c|c|c}
\text{7} & \text{00000111} & \text{becomes} \\
\hline
\text{-5} & \text{-0000101} & \text{+1111011} \\
\hline
\text{2} & \text{00000111} & \text{= 2}
\end{array}
\]

**Representation of Floating Point Numbers (Used For Reals)**

A floating point number may be represented by 3 parts in a single computer word.

\[ m \times b^e \]

where \( m \) is the mantissa and \( 0 \leq \text{mantissa} < 1 \)

\( b \) is the base

\( e \) is the exponent (the power of the base as a positive or negative integer)

eg 67.79 \( \times \) 10\(^{-3} \)

**Normalised notation** has no significant digits before the decimal point, multiplied by the base (10) to the appropriate power. Floating point numbers are stored in normalised form as this is the most efficient way.

eg 0.6779 \( \times \) 10\(^{-1} \)

**Some examples of floating point representation**

There are many possible ways of storing real numbers, and here are some examples.

**Example 1 – 16 bit word with 5 bit 2's complement exponent and normalised mantissa**

\[
\begin{array}{c|c|c}
\text{15} & \text{bits 14 - 10} & \text{bits 9 - 0} \\
\hline
\text{sign} & \text{exponent in 2s} & \text{Normalised mantissa} \\
\text{0 (+)} & \text{complement to base 2} & \\
\text{1 (-)} & \\
\hline
\end{array}
\]

Largest number that can be represented: 0 01111 1111111111 \( \rightarrow \) 0.1111111111 \( \times \) 2\(^{1111} \)

Smallest positive number that can be represented: 0 10000 1000000000 \( \rightarrow \) 0.1 \( \times \) 2\(^{-10000} \)

eg1 0 00001 1000000000

\[
\begin{align*}
\text{sign} & = \text{plus} \\
+ & 0.5 \times 2^1 = 1
\end{align*}
\]

eg2 1 00000 1101000000

\[
\begin{align*}
\text{sign} & = \text{minus} \\
+ & 0.1101 \times 2^0 = 0.8125
\end{align*}
\]

**Example 2 – IEEE 754 Floating Point**

The internal binary format used for floating point and double numbers. The format assigns a meaning to every possible combination of bits. There are also representation for NaN (Not a Number) and plus and minus infinity.

<table>
<thead>
<tr>
<th>Floating Point Bit Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>---------</td>
</tr>
</tbody>
</table>

Page 25
Bytes | Bits | Double | 8 bytes | 64 bits | 14 to 15 significant digits | ±4.94065645841246544e-324 to ±1.79769313486231570e+308 | is formed of 3 fields: 1-bit sign, 11-bit base 2 exponent, 52-bit fraction, lead 1 implied | a double can exactly represent integers in the range $-2^{53}$ to $+2^{53}$.

Bytes | Bits | Float | 4 bytes | 32 bits | 6 to 7 significant digits | ±1.40129846432481707e-45 to ±3.40282346638528860e+38 | is formed of 3 fields: 1-bit sign, 8-bit base 2 exponent, 23-bit fraction, lead 1 implied | a float can exactly represent integers in the range $-2^{24}$ to $+2^{24}$.

**Representation of Characters and Strings**

Computer memory is arranged in **words**. Words are groups of **bytes** (ie 8 bits = 1 byte) or a number of memory locations each able to hold 1 bit (binary digit). Characters are stored in the computer by using a code for each character. There are a number of standard binary codes used in computers to represent characters. ASCII and Unicode are common.

**ASCII**

American Standard Code for Information Interchange is the code commonly used with PCs or microcomputers. Initially the code used 7 bits but changed to 8 bits to allow up to 256 characters. ASCII needs 8 bits to represent one character. In both cases an extra **parity bit** would be added. (see below)

**Unicode**

This is the character set Java uses. Unicode includes the standard ASCII character set as its first 256 characters (with the high byte set to 0), but also includes several thousand other characters representing most international alphabets. To represent the **primitive** java type `char`, Unicode uses the 16 bit code for alphanumerics. The numerical values are unsigned 16 bit values between 0 and 65535. Unicode allows 16 bits or 2 bytes for each character (in total can represent 65536 character combinations) and so takes up twice as much memory for data storage as ASCII. In order to find the character connected to a particular unicode we use in Java the following:

```java
char characterVariable = (char) integer_value
int k = 65;
char c = (char) k;
```

This will cause `c` to take the value ‘A’, since ‘A’ is the character with unicode and ASCII code equal to 65.

A table of the ASCII codes is on the next page.

**Representation of Boolean**

A boolean variable has the two possible values **True** and **False**. This means a boolean value can be represented by one bit which has two values 0 and 1. In Java 1 represents **True** and 0 represents **False**. Although only one bit is required a particular implementation may use a byte to store the 1 or 0. In this case 8 bits would be used to store the boolean values.
<table>
<thead>
<tr>
<th>Dec Char</th>
<th>Binary</th>
<th>Dec Char</th>
<th>Binary</th>
<th>Dec Char</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>58</td>
<td>:</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>59</td>
<td>;</td>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>60</td>
<td>&lt;</td>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>61</td>
<td>=</td>
<td>120</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>62</td>
<td>&gt;</td>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>63</td>
<td>?</td>
<td>122</td>
<td>z</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>64</td>
<td>@</td>
<td>123</td>
<td>{</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>65</td>
<td>A</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>66</td>
<td>B</td>
<td>125</td>
<td>)</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>67</td>
<td>C</td>
<td>126</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
<td>68</td>
<td>D</td>
<td>127</td>
<td>DEL</td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
<td>69</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
<td>70</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
<td>71</td>
<td>G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
<td>72</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>73</td>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>DLE</td>
<td>74</td>
<td>J</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>DC1</td>
<td>75</td>
<td>K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>DC2</td>
<td>76</td>
<td>L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>DC3</td>
<td>77</td>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>DC4</td>
<td>78</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>NAK</td>
<td>79</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>SYN</td>
<td>80</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>ETB</td>
<td>81</td>
<td>Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>CAN</td>
<td>82</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>EM</td>
<td>83</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>SUB</td>
<td>84</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>ESC</td>
<td>85</td>
<td>U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>FS</td>
<td>86</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>GS</td>
<td>87</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>RS</td>
<td>88</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>US</td>
<td>89</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>SPACE</td>
<td>90</td>
<td>Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>91</td>
<td>[</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>92</td>
<td>\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>93</td>
<td>]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>$</td>
<td>94</td>
<td>^</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>%</td>
<td>95</td>
<td>_</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>&amp;</td>
<td>96</td>
<td>`</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>'</td>
<td>97</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>98</td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>)</td>
<td>99</td>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>*</td>
<td>100</td>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>+</td>
<td>101</td>
<td>e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>,</td>
<td>102</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>-</td>
<td>103</td>
<td>g</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>.</td>
<td>104</td>
<td>h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>/</td>
<td>105</td>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>106</td>
<td>j</td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>1</td>
<td>107</td>
<td>k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>108</td>
<td>l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>3</td>
<td>109</td>
<td>m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>4</td>
<td>110</td>
<td>n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>5</td>
<td>111</td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>6</td>
<td>112</td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>7</td>
<td>113</td>
<td>q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>8</td>
<td>114</td>
<td>r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>9</td>
<td>115</td>
<td>s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>:</td>
<td>116</td>
<td>t</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consider the following definition for a single dimensional array:

```java
int[] data = new int[2]; data[0] = 0; data[1] = 0; data[2] = 0;
```

This can be diagrammatically represented as follows (assuming array has been initialised to all zeroes):

Consider the following definition for a two dimensional array:

```java
int[][] data = new int[3][2];
```

This can be diagrammatically represented as follows (assuming array has been initialised to all zeroes):
**Data Compression** (Source Wikipedia)
In computer science and information theory, **data compression** or **source coding** is the process of encoding information using fewer bits (or other information-bearing units) than an unencoded representation would use through use of specific encoding schemes.

**Lossless** compression schemes are reversible so that the original data can be reconstructed, while **lossy** schemes accept some loss of data in order to achieve higher compression.

Lossless compression algorithms usually exploit statistical redundancy in such a way as to represent the sender's data more concisely without error. Lossless compression is possible because most real-world data has *statistical redundancy*. For example, in English text, the letter 'e' is much more common than the letter 'z', and the probability that the letter 'q' will be followed by the letter 'z' is very small.

Lossy data compression is possible if some loss of fidelity is acceptable. Generally, a lossy data compression will be guided by research on how people perceive the data in question. For example, the human eye is more sensitive to subtle variations in *luminance* than it is to variations in color. JPEG image compression works in part by "rounding off" some of this less-important information. Lossy data compression provides a way to obtain the best fidelity for a given amount of compression. In some cases, *transparent* (unnoticeable) compression is desired; in other cases, fidelity is sacrificed to reduce the amount of data as much as possible.

**Still Images**
One way to represent a picture is to consider the picture as a collection of dots called pixels (picture elements). In this representation the picture can be stored as a long string of bits representing rows of pixels in the picture. Each bit (0 or 1) corresponds to whether the pixel is black or white. Black and white images were economical because they only needed one bit per pixel.

If a picture has a resolution of 1000 x 5000 then there are 1000 pixels across and 5000 going down. eg a 1000 x 5000 image requires 625000 bytes (5,000,000 bits/8).

Colour pictures need a number of bits to represent each possible colour. Colour depth is the number of separate colours that a graphics file can handle.

Early images were then stored so only the best sixteen colours of an image were used. In this case 4 bits were needed to store information about each pixel. With the introduction of 256 colours images kept getting bigger.

**Audio**
Sampling is the process of turning analogue sound waves into digital (binary) signals. The system samples the sound by taking snapshots of its frequency and amplitude at regular intervals. The higher the sample rate the more accurate the digital sound to its real-life source and the larger the disk space required to record it.

Original audio adapters provided 8-bit audio where 8 bits were used to digitise each sound sample. This meant that there were 256 possible digital values which is usually adequate for speech but not for music.

Basically sampling should happen at twice the highest frequency to be recorded. A sound sampled at 44 KHz requires as much as 10.5 Mb of disk space per minute.

Many audio adapters include their own data-compression capability. One such algorithm is Adaptive Differential Pulse Code Modulation which reduces file size by up to 50% but loses sound quality.
Logic Circuits

The basic gates are as follows:

<table>
<thead>
<tr>
<th>Function / Gate</th>
<th>Symbol</th>
<th>Truth Table</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| **AND**         | ![AND symbol] | \[
| A & B f
| 0 0 0
| 0 1 0
| 1 0 0
| 1 1 1
| = 1 if both A and B are 1
| = 0 otherwise |
| **OR**          | ![OR symbol] | \[
| A ∨ B f
| 0 0 0
| 0 1 1
| 1 0 1
| 1 1 1
| = 0 if both A and B are 0
| = 1 otherwise |
| **NOT**         | ![NOT symbol] | \[
| A f
| 0 1
| 1 0
| = 0 if A is 1
| = 1 if A is 0 |

From these basic gates the following gates can be built:

<table>
<thead>
<tr>
<th>Function / Gate</th>
<th>Symbol</th>
<th>Truth Table</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| **NAND** (NOT AND)   | ![NAND symbol] | \[
| A & B f
| 0 0 1
| 0 1 1
| 1 0 1
| 1 1 0
| = 0 if A both and B are 1
| = 1 otherwise |
| **NOR** (NOT OR)     | ![NOR symbol] | \[
| A ∨ B f
| 0 0 1
| 0 1 0
| 1 0 0
| 1 1 0
| = 1 if both A and B are 0
| = 0 otherwise |
| **XOR** (Exclusive OR) | ![XOR symbol] | \[
| A ⊕ B f
| 0 0 0
| 0 1 1
| 1 0 1
| 1 1 0
| = 1 when A or B is 1
| = 0 when A and B are the same (ie both 0 or both 1) |
## List of Logic Laws

**Commutative Laws**

<table>
<thead>
<tr>
<th>L1</th>
<th>$a \land b \equiv b \land a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2</td>
<td>$a \lor b \equiv b \lor a$</td>
</tr>
</tbody>
</table>

**Laws of the Constants**

<table>
<thead>
<tr>
<th>L21</th>
<th>$\neg T \equiv F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L22</td>
<td>$\neg F \equiv T$</td>
</tr>
</tbody>
</table>

**Associative Laws**

<table>
<thead>
<tr>
<th>L4</th>
<th>$a \land (b \land c) \equiv (a \land b) \land c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L5</td>
<td>$a \lor (b \lor c) \equiv (a \lor b) \lor c$</td>
</tr>
</tbody>
</table>

**Law of Negation**

| L6  | $\neg \neg a \equiv a$ |

**Complement Rules of $\land$ and $\lor$**

<table>
<thead>
<tr>
<th>L7</th>
<th>$a \lor (b \land \neg a) \equiv a \lor b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L8</td>
<td>$a \land (b \lor \neg a) \equiv a \land b$</td>
</tr>
</tbody>
</table>

**Distribution Laws**

<table>
<thead>
<tr>
<th>L10</th>
<th>$\neg (a \lor b) \equiv \neg a \land \neg b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L11</td>
<td>$\neg (a \land b) \equiv \neg a \lor \neg b$</td>
</tr>
</tbody>
</table>

**De Morgan's Laws**

<table>
<thead>
<tr>
<th>L12</th>
<th>$a \land a \equiv a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L13</td>
<td>$a \lor a \equiv a$</td>
</tr>
</tbody>
</table>

**Absorption Laws**

<table>
<thead>
<tr>
<th>L33</th>
<th>$a \land (a \lor b) \equiv a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L34</td>
<td>$a \lor (a \land b) \equiv a$</td>
</tr>
</tbody>
</table>

**Idempotent Laws**

| L14 | $a \lor \neg a \equiv T$ |

**Law of the Excluded Middle**

| L15 | $a \land \neg a \equiv F$ |

**Law of Contradiction**
**Karnaugh Maps**

A **Karnaugh map (K-map)** is a graphical method of simplifying a Boolean logic expression. K-map can be used to simplify expressions involving 2, 3 or 4 Boolean variables.

**Drawing K Maps**

A K-maps is a pictorial arrangement of a truth tables and easier way to determine the minimum number of terms needed to express the function algebraically.

**Minterms** are formed by ANDing the variables in the table together. For example the truth table shown below contains 4 minterms. When expressed in a table, a function of \( n \) variables will have \( 2^n \) minterms.

Every function can be written as a **sum of minterms**.

To find the sum of midterms function:

1. **pick the rows in the table where the output is 1**
2. **OR these minterms together**.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Minterms</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( \sim A \land \sim B )</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( \sim A \land B )</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( A \land \sim B )</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>( A \land B )</td>
<td>1</td>
</tr>
</tbody>
</table>

Output = 1. Minterm for this row is \( (\sim A \land B) \lor (A \land B) \)

Adding the minterms in this way will always give you a function to describe the relationship between \( A \) and \( B \). However it will not always give you the **simplest** function. Arranging the minterms into a K-map allows us to look for patterns we may not otherwise see and find a simpler function to describe the relationship.

**Truth table**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Equivalent K-map**

![K-map diagram]

---

Page 32
Simplifying Expressions with two variables using K-maps

When using K-maps to simplify an expression we look for the 1’s in the table and group them with a rectangle. Groupings can contain 1, 2 or 4 terms and must never contain a 0!

Example 1

Truth table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ f = (A \land \neg B) \lor (A \land B) \]

Equivalent K-map

<table>
<thead>
<tr>
<th>B</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Minterms adjacent in bottom row
⇒ sum of minterms can be simplified to \( A \)
\[ f = (A \land \neg B) \lor (A \land B) = A \]

Example 2

Truth table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ f = (\neg A \land B) \lor (A \land \neg B) \]

Equivalent K-map

<table>
<thead>
<tr>
<th>B</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Selected minterms are not adjacent in the same row or column ⇒ no simplification possible
\[ f = (\neg A \land B) \lor (A \land \neg B) \]

Example 3

Truth table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ f = (\neg A \land B) \lor (A \land \neg B) \lor (A \land B) \]

Equivalent K-map

<table>
<thead>
<tr>
<th>B</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

In this case there are two groups of 1’s we can select.

Overlapping groups are allowed in K-map simplification.

Min terms sum \((A \land \neg B) \lor (A \land B)\) can be simplified to \(A\).
Min terms sum \((\neg A \land B) \lor (A \land B)\) can be simplified to \(B\).
\[ \Rightarrow f = (\neg A \land B) \lor (A \land \neg B) \lor (A \land B) = A \lor B \]

Note: an OR is placed between the two simplified expressions to form the final simplified solution.
Simplifying Expressions with three variables using K-maps

Groupings can contain 1, 2, 4 or 8 terms and must never contain a 0!

**Example 4**
K-map

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Sum of minterms = (¬A ∧ ¬B ∧ C) ∨ (A ∧ ¬B ∧ C)
Selected minterms are both in the ¬B and the C column so the expression can be simplified to:

\[ f = ¬B ∧ C \]

**Example 5**
K-map

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Sum of minterms
\[ f = (¬A ∧ ¬B ∧ C) ∨ (A ∧ ¬B ∧ C) ∨ (¬A ∧ B ∧ C) ∨ (A ∧ B ∧ C) \]
Selected minterms are all in both the C columns so the expression can be simplified to:

\[ f = C \]

**Example 6**
K-map

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Sum of minterms
\[ f = (¬A ∧ ¬B ∧ ¬C) ∨ (A ∧ ¬B ∧ ¬C) ∨ (A ∧ B ∧ ¬C) \]
Selected minterms can be simplified to:

\[ f = (¬B ∧ ¬C) ∨ (A ∧ ¬C) \]

**Note:** There are a couple of ways to tackle example 6.

(i) Think of the left and right edges of our table as ‘glued’ together to form a cylinder.

The minterms (A ∧ ¬B ∧ ¬C) and (A ∧ B ∧ ¬C) are now adjacent and can be summed as shown in the table above.

(ii) The table can be rearranged by sliding all columns to the right and forming the new table shown below.

The adjacent cells are now obvious.
Example 7

K-map

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Sum of minterms

\[ f = (A \land \neg B \land \neg C) \lor (A \land \neg B \land C) \lor (A \land B \land C) \]

**Three table cells cannot be selected together.**

There are two equally correct solutions for this problem.

Selected minterms as shown, can be simplified to:

\[ f = (A \land \neg B) \lor (A \land B \land C) \]

or

Grouping the right hand pair we have:

\[ f = (A \land \neg B \land \neg C) \lor (A \land C) \]

---

**Simplifying Expressions with four variables using K-maps**

Groupings can contain 1, 2, 4, 8 or 16 terms and must never contain a 0!

Example 8

Truth table

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Equivalent K-map

Using the minterms selected:

\[ f = (\neg A \land \neg B \land \neg C \land \neg D) \lor (A \land D) \lor (B \land C \land \neg D) \]
Example 9

The corners can be a little difficult to see how to group. Think about wrapping in two directions so that the corners come together in one block, almost like wrapping around a sphere.

Selected minterms can be simplified to:

$$f = (\neg B \land \neg D) \lor (B \land \neg C \land D)$$

**OR** the table can be rearranged by sliding all columns to the right and all rows down.

Selected minterms can be simplified to:

$$f = (\neg B \land \neg D) \lor (B \land \neg C \land D)$$

Example 10

Overlapping the groups gives the minimal number of groupings.

Selected minterms can be simplified to:

$$f = (B \land D) \lor (C \land D)$$

Example 11

Selected minterms can be simplified to:

$$f = D$$
Example: Design a logic circuit that will take a binary number in the range 0 - 3 and produce as output three times the input in unsigned binary representation.

Let the input bits be \( i_1 \) and \( i_0 \) and the output bits be \( o_3, o_2, o_1 \) and \( o_0 \)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i_1 )</td>
<td>( i_0 )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The black box (machine) we need will have 2 inputs and 4 outputs.

As there are 4 outputs, each with their own Boolean expression, you need 4 expressions.

\[
\begin{align*}
\text{o}_3 & = \text{i}_0 \land \text{i}_1 \\
\text{o}_2 & = \sim \text{i}_0 \land \text{i}_1 \\
\text{o}_1 & = \text{i}_0 \land \sim \text{i}_1 \lor \sim \text{i}_0 \land \text{i}_1
\end{align*}
\]

Thus the circuit would be:
The TOY machine has 256 words of main memory, 16 registers, and 16-bit instructions. There are 16 different instruction types; each one is designated by one of the opcodes 0 through F. Each instruction manipulates the contents of memory, registers, or the program counter in a completely specified manner. The 16 TOY instructions are organized into three categories: arithmetic-logic, transfer between memory and registers, and flow control.

Instruction Representation – main memory only

Summary of TOY instructions:

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>DESCRIPTION</th>
<th>FORMAT</th>
<th>PSEUDOCODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>halt</td>
<td>-</td>
<td>exit</td>
</tr>
<tr>
<td>1</td>
<td>add</td>
<td>1</td>
<td>R[d] ← R[s] + R[t]</td>
</tr>
<tr>
<td>2</td>
<td>subtract</td>
<td>1</td>
<td>R[d] ← R[s] - R[t]</td>
</tr>
<tr>
<td>3</td>
<td>and</td>
<td>1</td>
<td>R[d] ← R[s] &amp; R[t]</td>
</tr>
<tr>
<td>4</td>
<td>xor</td>
<td>1</td>
<td>R[d] ← R[s] ^ R[t]</td>
</tr>
<tr>
<td>5</td>
<td>left shift</td>
<td>1</td>
<td>R[d] ← R[s] &lt;&lt; R[t]</td>
</tr>
<tr>
<td>6</td>
<td>right shift</td>
<td>1</td>
<td>R[d] ← R[s] &gt;&gt; R[t]</td>
</tr>
<tr>
<td>7</td>
<td>load address</td>
<td>2</td>
<td>R[d] ← addr</td>
</tr>
<tr>
<td>8</td>
<td>Load</td>
<td>2</td>
<td>R[d] ← mem[addr]</td>
</tr>
<tr>
<td>9</td>
<td>store</td>
<td>2</td>
<td>mem[addr] ← R[d]</td>
</tr>
<tr>
<td>A</td>
<td>load indirect</td>
<td>1</td>
<td>R[d] ← mem[R[t]]</td>
</tr>
<tr>
<td>B</td>
<td>store indirect</td>
<td>1</td>
<td>mem[R[t]] ← R[d]</td>
</tr>
<tr>
<td>C</td>
<td>branch zero</td>
<td>2</td>
<td>if (R[d] == 0) pc ← addr</td>
</tr>
<tr>
<td>D</td>
<td>branch positive</td>
<td>2</td>
<td>if (R[d] &gt; 0) pc ← addr</td>
</tr>
<tr>
<td>E</td>
<td>jump register</td>
<td>-</td>
<td>pc ← R[d]</td>
</tr>
<tr>
<td>F</td>
<td>jump and link</td>
<td>2</td>
<td>R[d] ← pc; pc ← addr</td>
</tr>
</tbody>
</table>

Each TOY instruction consists of 4 hex digits (16 bits). The leading (left-most) hex digit encodes one of the 16 opcodes. The second (from the left) hex digit refers to one of the 16 registers, which we call the destination register and denote by d. The interpretation of the two rightmost hex digits depends on the opcode. Each opcode has a unique format.

With Format 1 opcodes, the third and fourth hex digits are each interpreted as the index of a register, which we call the two source registers and denote by s and t.

With Format 2 opcodes, the third and fourth hex digits (the rightmost 8 bits) are interpreted as a memory address, which we denote by addr.
Note: two instructions have no format listed ie. 0 and E. Consider the following examples.
Instructions 0000, 0A14, 0BC7 would all halt execution. The last 3 hex digits are ignored.
Instructions E500, E514, E5C7 would all change the value of the program counter to the
contents of register 5. The last 2 hex digits are ignored.

Toy Programming and Java

Variables – use memory locations 00 to 0F for variables and constants. Default data type is short integer.

Example 1: Assignment – x=3;  // x is stored in memory location 5

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Contents</th>
<th>Pseudocode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0003</td>
<td>data</td>
<td>The integer 3 is in memory address 00</td>
</tr>
<tr>
<td>05</td>
<td>0000</td>
<td>data</td>
<td>Location chosen for variable x</td>
</tr>
<tr>
<td>10</td>
<td>8A00</td>
<td>R[A] ← mem[00]</td>
<td>The integer 3 is copied to register A</td>
</tr>
<tr>
<td>11</td>
<td>9A05</td>
<td>mem[05] ← R[A]</td>
<td>Contents of register A stored in x</td>
</tr>
<tr>
<td>12</td>
<td>0000</td>
<td>halt</td>
<td>Stop execution</td>
</tr>
</tbody>
</table>

Example 2: Addition – x=6+8;  // x is stored in memory location 5

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Contents</th>
<th>Pseudocode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0006</td>
<td>data</td>
<td>The integer 6 is in memory address 00</td>
</tr>
<tr>
<td>01</td>
<td>0008</td>
<td>data</td>
<td>The integer 8 is in memory address 01</td>
</tr>
<tr>
<td>05</td>
<td>0000</td>
<td>data</td>
<td>Location chosen for variable x</td>
</tr>
<tr>
<td>10</td>
<td>8A00</td>
<td>R[A] ← mem[00]</td>
<td>The integer 6 is copied to register A</td>
</tr>
<tr>
<td>11</td>
<td>8B01</td>
<td>R[B] ← mem[01]</td>
<td>The integer 8 is copied to register B</td>
</tr>
<tr>
<td>12</td>
<td>1CAB</td>
<td>R[C] ← R[A] + R[B]</td>
<td>Integers 6 and 8 are added and the result (14) stored in register C</td>
</tr>
<tr>
<td>13</td>
<td>9C05</td>
<td>mem[05] ← R[C]</td>
<td>The integer 14 is copied to x</td>
</tr>
<tr>
<td>14</td>
<td>0000</td>
<td>halt</td>
<td>Stop execution</td>
</tr>
</tbody>
</table>

Multiplication can be performed by either

left shifts (each shift multiplies by 2), eg x=x * 10; would be implemented as x = x * 8 + x * 2;

or by adding the first number to itself, eg x = x * 10; would be implemented as x =
x+x+x+x+x+x+x+x+x+x;

(Note division can use right shifts or repeated subtraction)
Example 3: Multiplication – x = x * 8; // x is stored in memory location 5

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Contents</th>
<th>Pseudocode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0003</td>
<td>data</td>
<td>Number of places to left shift by</td>
</tr>
<tr>
<td>05</td>
<td>0004</td>
<td>data</td>
<td>Location chosen for variable x (with value 4 in it)</td>
</tr>
<tr>
<td>10</td>
<td>8A05</td>
<td>R[A] ← mem[05]</td>
<td>Load x (value 4) into register A</td>
</tr>
<tr>
<td>11</td>
<td>8B01</td>
<td>R[B] ← mem[01]</td>
<td>Load number of places to shift left (3) in register B</td>
</tr>
<tr>
<td>12</td>
<td>5AAB</td>
<td>R[A] ← R[A] &lt;&lt; R[B]</td>
<td>Register A is shifted left 3 places (becomes 24)</td>
</tr>
<tr>
<td>13</td>
<td>9A05</td>
<td>mem[05] ← R[A]</td>
<td>Contents of register A stored in x (24)</td>
</tr>
<tr>
<td>14</td>
<td>0000</td>
<td>halt</td>
<td>Stop execution</td>
</tr>
</tbody>
</table>

The if statement

Requires jumping over blocks of memory used for the condition true and condition false cases.

<table>
<thead>
<tr>
<th>Java</th>
<th>TOY Processing Sequence</th>
<th>Example 4</th>
</tr>
</thead>
</table>
| if (a==b) code1 else code2 | code2 if (a-b=0) branch to code1 op branch to cont code1 op | if (x==3)
| code2 op cont op | x=x+1; else x=x-1; |

Example 4: If statement

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Contents</th>
<th>Pseudocode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0000</td>
<td>data</td>
<td>variable x</td>
</tr>
<tr>
<td>01</td>
<td>0003</td>
<td>data</td>
<td>constant 3</td>
</tr>
<tr>
<td>02</td>
<td>0001</td>
<td>data</td>
<td>constant 1</td>
</tr>
<tr>
<td>10</td>
<td>8A00</td>
<td>R[A] ← mem[00]</td>
<td>load x into register A</td>
</tr>
<tr>
<td>11</td>
<td>8B01</td>
<td>R[B] ← mem[01]</td>
<td>load constant 3 into register B</td>
</tr>
<tr>
<td>12</td>
<td>2CAB</td>
<td>R[C] ← R[A] - R[B]</td>
<td>form test for x-3==0</td>
</tr>
<tr>
<td>13</td>
<td>CC18</td>
<td>if (R[C]==0) PC ← 18</td>
<td>if x==3 goto 18 (code1)</td>
</tr>
<tr>
<td>14</td>
<td>8B02</td>
<td>R[B] ← mem[02]</td>
<td>code2 : load constant 1 into register B</td>
</tr>
<tr>
<td>15</td>
<td>2CAB</td>
<td>R[C] ← R[A] – R[B]</td>
<td>take 1 off x and store result in register C</td>
</tr>
<tr>
<td>16</td>
<td>9C00</td>
<td>mem[00] ← R[C]</td>
<td>store register C in x (ie x=x-1)</td>
</tr>
<tr>
<td>17</td>
<td>C01B</td>
<td>if (R[0]==0) PC ← 1B</td>
<td>goto 1B (cont)</td>
</tr>
<tr>
<td>18</td>
<td>8B01</td>
<td>R[B] ← mem[01]</td>
<td>code1 : load constant 3 into register B</td>
</tr>
<tr>
<td>19</td>
<td>1CAB</td>
<td>R[C] ← R[A] + R[B]</td>
<td>add 3 to x and store result in register C</td>
</tr>
<tr>
<td>1A</td>
<td>9C00</td>
<td>mem[00] ← R[C]</td>
<td>store register C in x (ie x=x+3)</td>
</tr>
<tr>
<td>1B</td>
<td>0000</td>
<td>halt</td>
<td>cont – program continues here after if</td>
</tr>
</tbody>
</table>
While and for loops

Similar to if statement (to exit loop). For loops can be implemented as while loops

<table>
<thead>
<tr>
<th>Java while loop</th>
<th>Java for loop</th>
<th>Example 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x=0;</code></td>
<td><code>for (int x=0; x&lt;5; x++)</code></td>
<td><strong>start</strong></td>
</tr>
<tr>
<td><code>while (x&lt;5)</code></td>
<td></td>
<td><strong>here</strong></td>
</tr>
<tr>
<td><code>{</code></td>
<td><code>{</code></td>
<td><strong>if (x-5 &gt; 0) goto next</strong></td>
</tr>
<tr>
<td><code>x = x + 1;</code></td>
<td><code>}</code></td>
<td><strong>x = x + 1</strong></td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
<td><strong>goto here</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>next</strong></td>
</tr>
</tbody>
</table>

**Example 5: while / for loop**

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Contents</th>
<th>Pseudocode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0000</td>
<td>data</td>
<td>variable x</td>
</tr>
<tr>
<td>01</td>
<td>0005</td>
<td>data</td>
<td>constant 5</td>
</tr>
<tr>
<td>02</td>
<td>0001</td>
<td>data</td>
<td>constant 1</td>
</tr>
<tr>
<td>10</td>
<td>9000</td>
<td>mem[00] ← R[0]</td>
<td>start : x=0;</td>
</tr>
<tr>
<td>11</td>
<td>8A00</td>
<td>R[A] ← mem[00]</td>
<td>load x into register A</td>
</tr>
<tr>
<td>12</td>
<td>8B01</td>
<td>R[B] ← mem[01]</td>
<td>load constant 5 into register B</td>
</tr>
<tr>
<td>14</td>
<td>DC18</td>
<td>if (R[C]&gt;0) PC ← 18</td>
<td>goto 18 (next) if loop finished</td>
</tr>
<tr>
<td>15</td>
<td>8D02</td>
<td>R[D] ← mem[02]</td>
<td>load constant 1 into register D (to add to x)</td>
</tr>
<tr>
<td>16</td>
<td>1AAD</td>
<td>R[A] ← R[A] + R[D]</td>
<td>store x+1 in register A</td>
</tr>
<tr>
<td>17</td>
<td>C011</td>
<td>if (R[0]==0) PC ← 11</td>
<td>goto 11 (here) to test for loop end</td>
</tr>
<tr>
<td>18</td>
<td>0000</td>
<td>halt</td>
<td>next : – program continues here after loop</td>
</tr>
</tbody>
</table>

**Structure of the CPU**

**Control Unit**
The control unit of the CPU contains circuitry that directs the entire computer system to carry out, or execute, stored program instructions. The control unit communicates with both the arithmetic/logic unit and main memory.

**Register**
Registers are temporary storage areas for instructions or data. They are not a part of memory. They are special additional storage locations that offer the advantage of speed. Registers work under the direction of the control unit to accept, hold, and transfer instructions or data.

**ALU**
Stands for Arithmetic/Logic Unit. This is the part that executes the computer's commands. A command must be either a basic arithmetic operation: `+ - * /` or one of the logical comparisons: `> < = not =`.

**Memory**
Memory is the part of the computer that holds data and instructions for processing. Although closely associated with the central processing unit, memory is separate from it.
**Bus**  
A bus is a collection of wires and connectors through which the data is transmitted. A bus is used to connect the components of the CPU and Memory. The bus has two parts -- an address bus and a data bus. The data bus transfers actual data whereas the address bus transfers information about the data and where it should go.

---

<table>
<thead>
<tr>
<th>The Machine Cycle</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fetch</strong> -</td>
<td>get an instruction from Memory and place in Instruction Register in Control Unit</td>
</tr>
<tr>
<td><strong>Decode</strong> -</td>
<td>converts instruction into computer commands that control the ALU and Memory</td>
</tr>
<tr>
<td><strong>Execute</strong> -</td>
<td>actually process the commands using the ALU and Memory</td>
</tr>
</tbody>
</table>