

## Program Structure

```
// this is my first C++ program

#include <iostream>
using namespace std;

int
main()
{
    cout << "Hello World" << endl;

    return (0);
}
```

- any line beginning with `//` is a comment: it is ignored by the computer but gives information to the programmer
- all of our programs will start with `#include <iostream>`
- all of our programs will include the line `using namespace std;`
- the main body of the program starts with `int main()` and finishes with `return (0);`
- all program statements must end with a semicolon `;`
- curly brackets `{ }` show the start and end of the main body of the program
- `cout` is the output channel: it sends text to the screen
- `endl` tells the computer to go onto a new line
- computers are very fussy: if a comma, bracket or semicolon is missing the program will not work

## CA146 Introduction to Programming in C++

```
#include <iostream>
using namespace std;

int
main()
{
    cout << "Hello World" << endl;
    cout << "Goodbye World" << endl;

    return (0);
}
```

This program prints two lines on the screen:

```
Hello World
Goodbye World
```

## Variables

```
#include <iostream>
using namespace std;

int
main()
{
    int x, y, z;

    x = 2;

    y = 3;

    z = x + y;

    cout << "the value of z is " << z << endl;

    return (0);
}
```

- variables are symbols for data
- variables must be defined before they can be used  
`int x, y, z;` defines three variables `x`, `y` and `z` as integers
- you can assign values to variables using the equals sign =  
`x = 2;` sets `x` equal to 2
- integers can also be zero or negative
- you can do arithmetic using variables  
`z = x+y;` sets `z` equal to the sum of the values of `x` and `y`
- `cout << z << endl;` prints the value of `z` on the screen
- the above program will print  
`the value of z is 5`

## Integer Arithmetic

```
#include <iostream>
using namespace std;

int
main()
{
    int x, y, z;
    int a, b, c;

    x = 10; y = 2;

    z = x - y;

    a = x * y;

    b = x / y;

    x = x + y;

    return (0);
}
```

- the four arithmetic operators are + - \* /
- it is legal to add a variable to itself

```
x = x + y;
```

- when one integer variable is divided by another the result is always an integer e.g.

```
x = 20;    y = 3;
z = x / y;
```

z will get the value 6 (NOT 6.6666)

## Floating Point Arithmetic

```
#include <iostream>
using namespace std;

int
main()
{
    float p, q, r;

    p = 3.5;    q = -0.3;

    r = p / q;

    cout << "the value of r is " << r << endl;
}
```

- floating point variables represent decimal numbers - they are defined by

```
float p, q, r;
```

- the above program would print

```
the value of r is -11.66666
```

unlike integer arithmetic the decimal part is retained

- division by zero generates an error

```
y = 0.0;
z = x / y;
```

would cause the computer to print

```
Floating Point Error
```

when the program is run.

## Data Input

```
#include <iostream>
using namespace std;

int
main()
{
    int x,y;

    cin >> x;

    y = x * x;

    cout << "the square is " << y << endl;

    return 0;
}
```

- `cin` is the input channel. It reads data from the keyboard
- the above program will wait until the user types a number into the keyboard
- it then prints out the square of that number

## Summary

- Program Structure

```
#include <iostream>
using namespace std;
```

```
int main()
{
    .
    .
    return (0);
}
```

- Input/Output

Data can be read in from the keyboard and out to the screen using `cin` and `cout`

```
cin >> x;
cout << "....." << endl;
```

- Variables

Variables store data. So far you have met integer and floating point variables.

```
int x,y;
float a,b;
```

- Arithmetic

You can carry out arithmetic on variables using operators `+` `-` `*` `/`

```
z = x + y;    a = b / c;
```

## Decisions

```
#include <iostream>
using namespace std;

int
main()
{
    int num1;

    cin  >> num1;
    if (num1 > 999) {
        cout << "greater than 999" << endl;
    }
    return (0);
}
```

- The `if` statement is used to test if expression between the round brackets (.....) is true.
- If the expression is true then the computer carries out the instructions between the next set of curly brackets {.....}
- This program waits for the user to input a number. If the number is greater than 999 the program prints this is greater than 999. Otherwise it just stops.
- The symbol `>` is a logical operator. Here are some more logical operators:
  - `<` less than
  - `>=` greater than or equal to
  - `<=` less than or equal to
  - `==` equal to
  - `!=` not equal to

NB. The double equals `==` has a different meaning from the single equals `=`

- For example the line `if (num1 == 999)` tests if `num1` is exactly equal to 999.

## Combining Two Logical Tests

```
#include <iostream>
using namespace std;

int
main()
{
    int num1;

    cin >> num1;

    if(num1 > 999 && num1 < 1999) {
        cout << "between 999 and 1999" << endl;
    }
}
```

- This program waits for the user to input a number and then tests to see if it lies between 999 and 1999
- An expression which uses the logical operator `&&` will only be true if both halves of the expression are true
- An expression which uses the logical operator `||` will be true if one or other or both of the halves of the expression are true

For example

```
if (num1 < 999 || num1 > 1999)
```

tests if num1 is less than 999 OR greater than 1999.

## Negation

- The logical operator ! negates the expression following it
- For example

```
if (! (num1 > 999) )
```

is true if num1 is NOT greater than 999

- Integer variables are also regarded as logical variables e.g.

```
if (num1)
```

is true if num1 is not zero

- `if (!num1)`

is true if num1 is equal to zero

**if ... else**

```
#include <iostream>
using namespace std;

int
main()
{
    int num1;

    cin >> num1;

    if (num1 > 0) {
        cout << "greater than zero" << endl;
    }
    else if (num1 == 0) {
        cout << "equal to zero" << endl;
    }
    else {
        cout << "less than zero" << endl;
    }
    return (0);
}
```

- This program waits for the user to input a number and then performs three different actions depending on the value of the number.
- The `if ... else` structure tests for conditions one after the other

**switch**

```
#include <iostream>
using namespace std;

int
main()
{
    int input_value;

    cout << "Enter a value (1-3)" << endl;
    cin >> input_value;

    switch(input_value)
    {
        case 1 : cout << "You chose 1" << endl;
                 break;
        case 2 : cout << "You chose 2" << endl;
                 break;
        case 3 : cout << "You chose 3" << endl;
                 break;
        default: cout << "Invalid " << endl;
    }
    return (0);
}
```

- This program prompts the user to input a integer from 1 to 3.
- It carries out a different action for each value of the input value
- The different possible value appear after the word `case`
- Each action is terminated by the word `break`
- There is a default action which is carried out if none of the above cases occurs
- The `switch` statement is often used to create menu driven programs

**while loops**

```

#include <iostream>
using namespace std;

int
main()
{
    int x, i;
    x = 1; i = 1;
    while (i < 10) {
        x = x * 2;
        cout << "2 to the power " << i << " = "
             << x << endl;
        i = i + 1;
    }
    return (0);
}

```

- the `while` statement tests if the condition (`i < 10`) is true. If so the commands in the curly brackets are executed and then the condition is tested again. This is called a loop because the same statements are repeated many times

- the output from this program would be:

```

2 to the power 1 = 2
2 to the power 2 = 4
2 to the power 3 = 8
2 to the power 4 = 16
2 to the power 5 = 32
2 to the power 6 = 64
2 to the power 7 = 128
2 to the power 8 = 256
2 to the power 9 = 512

```

- Each cycle of the loop is called an iteration. This loop went through nine iterations.

- The `while` statement tests for the condition at the *start* of each iteration

- At each iteration the previous value of `x` was multiplied by 2 and the value of `i` was incremented by 1.

**do ...while loops**

```
#include <iostream>
using namespace std;

int
main()
{
    int x, i;
    x = 1; i = 1;
    do {
        x = x * 2;
        cout << "2 to the power " << i << " = "
             << x << endl;
        i = i + 1;
    } while (i < 10)
    return (0);
}
```

- this program is the same as the previous program except that we have a `do while` loop rather than a `while` loop
- the `do while` statement tests for the condition at the end of an iteration rather than at the start
- therefore a `do while` loop will always have at least one iteration
- the output from this program would be the same as the previous one

## A Simple Menu Program

```
#include <iostream>
using namespace std;

int main()
{
    int choice;

    do {
        cout << "input a no. (1-4)" << endl;
        cin >> choice;
        switch (choice) {
            case 1: cout << "You chose 1" << endl;
                    break;
            case 2: cout << "You chose 2" << endl;
                    break;
            case 3: cout << "You chose 3" << endl;
                    break;
            case 4: cout << "Goodbye" << endl;
                    break;
            default: cout << "Invalid" << endl;
        }
    } while (choice != 4);

    return (0);
}
```

- This shows the use of `do while` and `switch` to create a menu program
- This program prompts for an input. If the input is 1, 2 or 3 it prints a line of text to the screen and then prompts again. If the input is 4 the program quits.
- Notice the nesting of the curly brackets - one pair of curly brackets nested inside another

## Summary

- `if .. else` structures can be used to carry out different actions depending on different logical conditions
- logical expressions can be created using logical operators
  - > greater than
  - < less than
  - >= greater than or equal to
  - <= less than or equal to
  - == equal to
  - != not equal to
  - && and
  - || or
- `switch` can be used to carry out different actions depending on the value of an integer variable
- `while` and `do while` loops can be used to repeat sets of commands over and over again while a certain condition is true