Control structures and Functions

Introduction

- Programs are not limited to a linear sequence of instructions. During execution a program may repeat code or make decisions.
- Conditional structures
- Iteration structures; loops

Conditional structure: if and else

- Used to execute instructions if a condition is fulfilled

if (condition) statement

- The (condition) is the expression that is being evaluated.
- If we could alternatively specify what we want to happen in the event the condition is not fulfilled

for e.g.

```
if (condition) {
statement1 ;
}
else { statement2;}
```

Iteration structures

- This repeat a statement a certain number of times while the condition is true
- **While loop**
- **While (expression) { statements; }**

Do while loop

- The format is do statement while (condition)
- The functionality is the same as the while loop

except for the fact that the statement is executed at least once regardless of whether the condition is true or not

For loop

- The format is:

for (initialization; condition;increase) statement

and its main function is to repeat statement while condition remains true, like the while loop. But in addition, the for loop provides specific locations to contain an initialization statement and an increase statement. So this loop is specially designed to perform a repetitive action with a counter which is initialized and increased on each iteration.

- initialization is executed. Generally it is an initial value setting for a counter variable. This is executed only once.
- condition is checked. If it is true the loop continues,
- Otherwise the loop ends and statement is skipped (not executed).
- Statement is executed. As usual, it can be either a single statement or a block enclosed in braces { }.
- Finally, whatever is specified in the increase field is executed and the loop gets back to step 2.

- The initialization and increase fields are optional however the semi-colons separating the three categories are mandatory.
- Using commas we could specify more than one condition for each category. For example
- for (n=0, i=100; n!=I; n++, i--)

Jump statements

- The break statement
- The continue statement
- The exit function

Break statement

- Using break we can leave a loop even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end. For example, we are going to stop the count down before its natural end (maybe because of an engine check failure?):

Continue statement

- The continue statement causes the program to skip the rest of the loop in the current iteration as if the end of the statement block had been reached, causing it to jump to the start of the following iteration.

Goto statement

- goto allows to make an absolute jump to another point in the program. You should use this feature with caution since its execution causes an unconditional jump ignoring any type of nesting limitations.
- The destination point is identified by a label, which is then used as an argument for the goto statement. A label is made of a valid identifier followed by a colon (:).

Exit function

- exit is a function defined in the cstdlib library.
- The purpose of exit is to terminate the current program with a specific exit code. Its prototype is:

void exit (int exitcode);

- The exit code is used by some operating systems and may be used by calling programs. By convention, an exit code of 0 means that the program finished normally and any other value means that some error or unexpected results happened.

Selective structures: Switch

- The syntax of the switch statement is a bit peculiar. Its objective is to check several possible constant values for an expression. Something similar to what we did at the beginning of this section with the concatenation of several if and else if instructions.

Switch

- Switch (expression)

{ case 1:

Statements;

Break;

Case 2: statements; break;

Default: statements;

}

Functions

- A function is a group of statements that is executed

when it is called from some point of the program.

- Functions assist us to structure our programs in a more modular form hence enabling us access all

the potential that structural programming can offer

us in C++

- The format of a function is

type function_name (parameter1, parameter2, ….) {statements;}

- Type: Data type specifier of the type returned by the function
- Name: the identifier that shall be used to call or refer to the function
- Parameters: Each consists of
  - Data type specifier
  - Identifier
  - E.g.: int x
- Statements: Function body

//function example

```
#include<iostream>
using namespace std;
int addition (int a, int b)
{
int r;
r=a+b;
return (r);
}
int main ()
{
int z;
z = addition (5,3);
cout << "The result is " << z;
return 0;
}
```

Functions

- Passing by value

- We have been passing by value e.g

```
int z,x=5,y=6;
z= addition(x,y);
```

- When the function above is called the value of the variables are passed to the function but not the variables themselves

```
int addition (int a, int b)
```

```
z = addition (  5,      3 )
```

- Any modifications to the variables a and b within the function will not have no any effect on x and y variables them selves.

Functions

- Passing by reference
- We need to modify the values of external variables within a function, we pass by reference.
- In this case we pass the variable itself but not a copy of the value

```
// passing parameters by reference
 #include <iostream>
using namespace std;
 void duplicate (int& a, int& b, int& c)
 {
 a*=2;
 b*=2;
 c*=2;
 }
int main ()
 {
 int x=1, y=3, z=7;
 duplicate (x, y, z);
 cout << "x=" << x << ", y=" << y << ", z=" << z;  // x=2, y=6, z=14
return 0;
 }
```

Functions

- We use the & to mean we are passing by reference not value.
- Any modification to local variables will have an effect if passing by reference (&) if used.
- Thus more than one value can be returned.
- E.g float subtract(float &a , float &b);

Functions

Default values

- Functions can be declared with default variables to cater for events when a function is called if left blank.
- An assignment operator (=) is used with a value in the function declaration.
- E.g int add (int a, int b, int c=4);
- If a value is specified, the default is ignored.

Function overloading

- C++ allows functions to share the same name but differ in types or number of parameters they have.
- E.g

```
int operate (int a) { return (a*2); }

int operate (int a, int b) { return (a+b); }

float operate (float a) {return (a/2); }
```

Function overloading

- The functions have the same names but the compiler is able to tell which one is being called by different types or parameters.
- A function can not only be overloaded by its return types, at least one of the parameters must be different.

Inline functions

- This tells the compiler that inline substitution is preferred. The code generated by the function body is inserted every time the function is called instead of a regular call to it.
- The function behavior does not change
- During the function call, keyword inline is not used
- Format:

inline type function_name(parameters,..){
statements;}

Function recursivity

- Functions can call themselves.
- It can be using in sorting and calculating factorial numbers.

```
// factorial calculator
#include <iostream>
 using namespace std;
 long factorial (long a) {
 if (a > 1){
 return (a * factorial (a-1));
}
 else { return (1); }
 int main () {
long number;
cout << "Please type a number: ";
 cin >> number;
cout << number << "! = " << factorial (number);
 return 0; }
```

Function declaration

- All the fucntions above were declared before the main function
- Function prototypes can be used to declare functions.
- A function declaration is shorter than a definition but enough for the compiler to determine the return types and the types of parameters.

type name (parameters,…);

Function declaration

- The declaration does not need the identifiers,
- Eg

int add (int a,int b);

is the same as

int add(int,int);