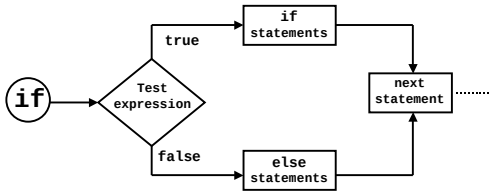
C++

Quick Overview

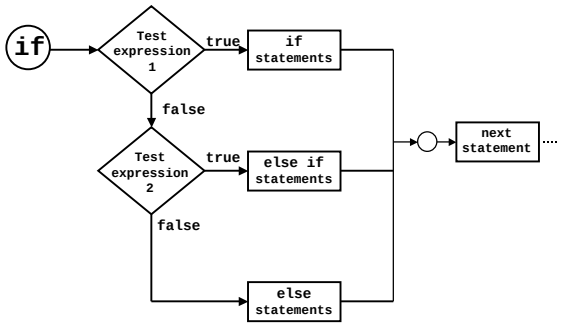
Conditional Execution: the `if` statement



Example: `if` statement

```
if (count) { //I.e. if count != 0
  cout << "Value is: "<< count<<"\n";
}
else{
  cout << "Now Zero\n";
}
```

Multiple Choice: `if/else if`

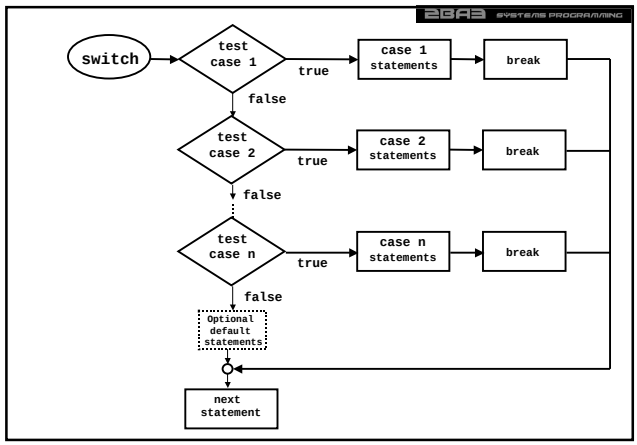


Example: `if/else if`

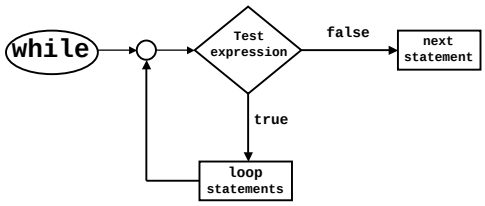
```
if (count > 50) {
  cout << "Value is greater than 50\n";
  do something…
}
else if (count >20{
  cout << "Value is between 20 and 50\n";
  do something…
}
else{
  cout << "Value is less than 20\n";
  do something…
}
```

Multiple Choice: the `switch` statement

```
switch( number ){
  case 1:
     cout << "One";
     break;
  case 2:
     cout << "Two";
     break;
  case 3:
     cout << "Three";
     break;
  default:
     cout << " Not one, two or three ";
}
cout << " \n ";
```



Repetition constructs: while loop



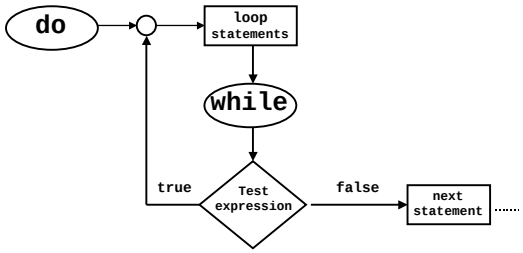
Example: while loop

while

```
count = 0;

while ( count > 0 ) {
  …Do something…
}
/* Loop never executed */
```

Repetition constructs: do while loop



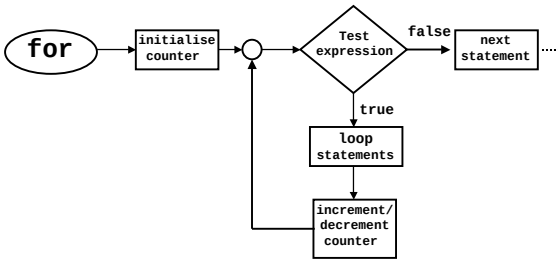
Example: do while loop

do while

```
countdown = 0;
do {
  …Do something…
  countdown--;
} while ( countdown > 0 );

/* Loop executed once */
```

Repetition constructs: for loop



Example: for loop

for

```
for (count = 10; count > 0;count--){
  // Do something
}

/*Loop executed exactly 10 times */
```

Console I/O

- In C++, I/O is performed using the << (insertion) and >> (extraction) operators
- Output:

```
cout << 42;
cerr << "Error in Data\n";
```

- Input:

```
int height;
cout << "Enter height: ";
cin >> height;
```

Fundamental Data Types in C++

char	Holds a character (min 8bits)
int	Machine word that holds an integer (min 16 bits)
unsigned int	(all integer types can be made unsigned)
short	Less precision than int (min 16 bits)
long	More precision than int (min 32 bits)
long long	More precision than int (min 64 bits
float	Held in floating point form
double	More precision than float

Data Types

- The exact representation or “range of values” that an item of type **int** can take, may well differ between implementations of C++.
- It all depends on the size in bytes of a machine word, on the computer used.
- C++ allows a variable to be initialized at the same time that it is declared, e.g:

```
int count = 0;
```

Typedef

- You may define new data types explicitly.
- You are not defining a new data class, but defining a new name for an existing type.
- This can make machine-dependent programs more portable
- e.g. if on one machine, an **int** is 2 words, and on another 4 words: you could define one of:

```
typedef int Integer, or
typedef long int Integer
```

- Then, instead of changing every declaration of **int** in your program, you only need to change the **typedef** statement

Constants

- Constants can be given a symbolic name to make a program more readable
- They are declared in the same way as variables, only no value may be assigned to them.

```
const   int MAX = 10;
const   float PI = 3.1415926;
```

Constants using #define

- Constants are also commonly defined using the preprocessor directive: **`#define`**

```
#define MAX 10
#define PI 3.1415926
```

- Note that this is not a C++ statement:
  - It is essentially a global replacement
  - Adding a semicolon ; will cause an error

Constants using #define

If you have **`#define MAX 10`** defined, and write the following:

```
for(int i=0;i<MAX;i++)…
```

the preprocessor will replace it with:

```
for(int i=0;i<10;i++)…
```

Can you see what will happen if you have:

```
#define MAX 10;
```

???

Enumerations

Variables may be restricted to have specific values as follows:

```
enum Color {RED,BLUE,GREEN};
Color car1, car2;
car1 = RED;
car2 = GREEN;
```


Arithmetic Operations

The arithmetic operators in C++ are:

- **+** Addition
- **-** Subtraction
- ***** Multiplication
- **/** Division
- **%** Modulus or Remainder

Examples:

- **5/2** → Integer Division (= 2)
- **5/2.0** → Floating point division (5.0/2.0 = 2.5)
- **5%2** → Modulus (= 1)

Relational Operators

- **==** → equal
- **!=** → not equal
- **<** → less than
- **>** → greater than
- **<=** → less than or equal
- **>=** → greater than or equal

Logical Operators

- **&&** → logical AND
- **||** → logical OR
- **!** → logical NOT

Classes

- A class is a way of encapsulating code and data which interact together into a single unit. This single unit can then be used to stamp out objects of a similar type.
- In C++ an object is an instance of a class. An object combines variables and the code which manipulates these variables into a single named item.

Example of A C++ class

```
class Circle {
public:
     float centreX;
     float centreY;
     float radius;
     int colour;
     Circle();    // constructor
     void Draw();
     ~Circle();   // destructor
};
```


Example: usage

Several circles may then be declared and drawn:

```
void main()
{
Circle circle1, circle2;

circle1.centreX = 1.0;
circle1.centreY = 1.0;
circle1.colour = 1;
circle1.radius = 2.0;
circle1.Draw();

circle2.centreX = 5.5;
…Do some more stuff…
Circle2.Draw();
}
```


Classes and Objects

- The class is a specification of an object.
- It is a description of storage combined with a description of the operations or methods that may be performed on this storage.
- An instance of the class, i.e. an object, must be created before any physical operations can be performed.
- Each physical representation will contain a copy of the data items that may be manipulated with the operations provided.
- These functions of the class are sometimes referred to as access routines or methods.

Functions

- The methods in the class are C++ functions
- A function is simply a grouping of code that can be executed to perform an action.
- For a method, this will usually involve the use of the data items declared within the class.
- The return type of the function is specified before the name.
- Parameters may be passed to a function, so that the values which the function operates on may be varied.

Returning results

- There can only be one single result from a function (or none).
- There are other ways of returning multiple items
  - using the parameter mechanism, to export values back to the calling environment.
- The return statement causes an immediate exit from the function.
- A function does not need to return a value explicitly, when the return type is void.

Visibility of Class members

The access rights to members of a class can be controlled by the visibility modifiers public and private:

```
class Account{
private:
   float  theBalance; //Not visible outside class
public:
   Account();          // Visible outside class
   float  AccountBalance();
   float  Withdraw(float money);
   void   Deposit(float money);
};
```

Separating Interface from Implementation

- It is good programming practice to make variables private to the class and to make public only those functions of the class which a user will use.
- The user of the class wants to know:
  - How to declare an instance of the class
  - What methods they can use to manipulate an instance of the class
- They do not want to know:
  - How any data items have been represented
  - The algorithms employed to implement the operations they have asked for

Local variables

- A variable declared inside a function only exists for the duration of that function.
- Space for the variables is created automatically on a run-time stack.
- When the function exits, the space is returned back to the system.
- It is good programming practice to only allow a function to access:
  - Parameters to a function
  - Local variables
  - Data items contained in the class of which this function is a member

Function prototypes

- In C++ all functions have to be specified before use.
- This is can done with a function prototype statement e.g.

```
float sum (float, float); //or....
float sum (float num1, float num2);
```

- If a function is a member of a class, then its specification is contained in the class specification.
- If a function does not declare a result then it is declared as a void function.
- Likewise, if a function has no parameters then void is used in place of any parameters. E.g. **`void draw(void)`**
- The use of void to indicate no parameters to a function is optional. Many people will just use an empty pair of braces. E.g. **`void draw()`**

Files

- It is possible to put your whole C++ program in one file
  - Just make sure everything is specified before use.
- But it is common practice to divide a C++ program into files
  - Speeds up recompilation if you only change one file
  - Makes it easier for multiple programmers to work on the same program
  - Helps make program more modular
  - Helps reuse of classes if each class is in a separate file

Files

- There are two types of files
- C++ code
  - In .cpp file
  - Contain C++ functions, global variables
  - Declarations of classes, functions, variables used only in this file
- Header (.h) files
  - Contain necessary declarations for the stuff in the C++ file to be used by other C++ files
  - Declarations of classes used by other C++ files
  - Declarations of functions used by other files

Files

- Typically, each C++ class has two files
- account.cpp account.h
- .h file contains declaration of class

```
class Account{
private:
   float  theBalance; //Not visible outside class
public:
   Account();          // Visible outside class
   float  AccountBalance();
   float  Withdraw(float money);
   void   Deposit(float money);
};
```

Files

- .cpp file contains implementations

```
#include "account.h"

float Account::AccountBalanace {
     return theBalance
};

…
```


Call by value

- Parameters are passed to a function in different ways depending on how the parameter is to be used:
  - Call by Value, or
  - Call by Reference
- Call by value: a value is imported into a function and no change occurs to the original variable in the calling function.

Call by value: Example

```
float sum(float a, float b){
         return a+b;
      }
```

usage:

```
num1 = 10;
num2 = 20;
answer = sum(num1, num2);
```


Call by reference

- Occasionally a value is required to be exported from the function; in this case a call by reference is used.
- call by reference does not copy the item, instead a reference to the item (contained in a machine word) is passed.

Call by reference

- A function **swap** to interchange the content of two variables requires data to be both imported and exported from the function.

```
void swap(int &a, int &b){
int temp;
temp = a;
a=b;
b=temp;
}
```


Const. Parameters

- To add extra security to functions the **const** specifier may be added to a parameter description to indicate that this parameter is read only inside the body of the function.
- If a write to this parameter is coded by a programmer, then a compile time error will be generated.

```
void wrong (const int item, const char& ch ){
     item = 123; // Both these statements will
     ch = 'a';   // fail at compile time
     }
```