C++ Basics

Introduction

C++ is almost a superset of the C programming language - generally a C code is valid in C++, but not always, since C++ is stricter than C in some respects (such as typechecking for function parameters)

The following is a sample code that works both in C and C++ :

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return(0);
}
```

1. `#include <stdio.h>` - instructs the compiler to include the declaration of the standard input/output library functions (which declares “printf” among other things)

2. This program defines a function called `main`. Every C or C++ program must have a function called `main` and the program starts by executing this function.

3. The body of this function `main` contains a call to `printf` function which writes `"Hello, World!\n"` to the standard output.

4. Backslash (“\”) followed by another character denotes a special character; in this case `\n` is a newline character.

5. `main` is of type `int` and returns the value `0` to the operating system.

The following program (valid for C++ only) produces the same result as the previous one.

```
#include <iostream.h>

int main() {
    cout << "Hello, World!\n";
    return(0);
}
```

1. This program uses “streams”, special classes used for input and output.
2. `cout` is the standard output stream.
3. The operator `<<` (“puts to”) writes its second argument into the first. In this case, the string `"Hello, World!\n"` is written onto the standard output stream `cout`.

Fundamental Types (Same for C and C++)

Keyword	**Explanation**
`char`	character (occupies 1 byte)
`int`	integer (occupies 1 word, 16-bit or 32-bit depending on the computer)
`long int`	integer (occupies 32 bits)
`float`	floating point number - single precision (6 digits)
`double`	floating point number - double precision (15 digits)

Example using basic types, if-statement and input from standard input (`cin`):

```
// Program that converts inch-to-centimeter
// and centimeters-to-inch

#include <iostream.h>

int main() {
    const float fac = 2.54;
    float x,in,cm;
    chat ch = ’y’;

    cout << "enter length:";

    cin >> x; // Read floating number
    cin >> ch; // Read suffix (one character)

    if  (ch == ’i’) { //inch
        in = x;
        cm = x*fac;
    }
    else  if (ch == ’c’) { //cm
        in = x/fac;
        cm = x;
    }
    else in=cm=0;

    cout << in << "in=" << cm << "cm\n";
    return 0;
}
```

Note in previous example:

1. `fac` and `ch` are initialized at the time of declaring.
2. `fac` is declared to be “constant” : attempting to change its value later would result in error.
3. `x` gets a floating number from the standard input stream `cin`. `ch` gets a character from `cin`.

Equivalent statement using the C library stdio.h are :

```
scanf("%f",&f);
scanf("%c",&ch);
```

- `%c` - special format indicating the type of the variable (in this case a `char`)
- `&ch` - “address of” variable `ch` must be given.

The if-statement could be replaced by a switch statement :

```
switch(ch) {
    case ‘i’ :  in = x;
                cm = x*fac;
                break;
    case ‘c’ :  in = x/fac;
                cm = x;
                break;
    default  :  in=cm=0;
}
```

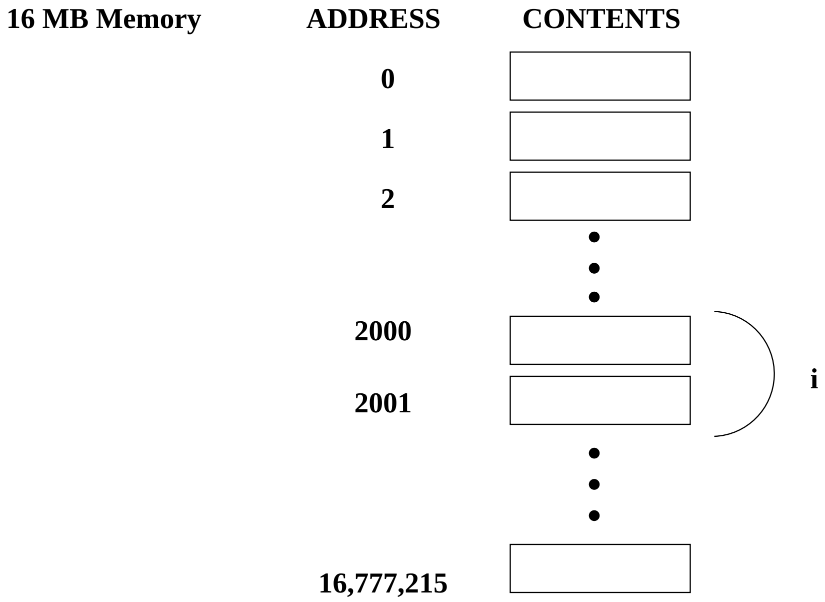
Derived Types (All exist in C except for reference)

- * - Pointer
- & - Reference
- ( ) - Function
- [ ] - Array

Pointer Variables *

- Each byte in memory has a unique address.
- A machine with 16 Megabytes of main memory has 16,777,216 of these bytes.
- Variables occupy one or more bytes of memory.

- In the following example, variable `i` occupies bytes 2000 and 2001, so `i`'s address is 2000.



A pointer is a variable that stores the address of another variable.

Example :

```
  ...
int i = 0;  // suppose i occupies positions 2000 and 2001
int *p;     // declares "pointer to" integer

p = &i;     // p gets the "address of" i (2000 in our example)
*p = 1;     // integer starting at memory pointed by p gets value 1

cout << "address is " << p << "\n";
cout << "value of i is " << i << "\n";
  ...
```

Output will be :

```
address is 2000;
value of i is 1;
```

Note : Here the symbol & is the unary operator “address of”; it is not a reference variable.

Reference Variable &

References are declared as synonyms to variables

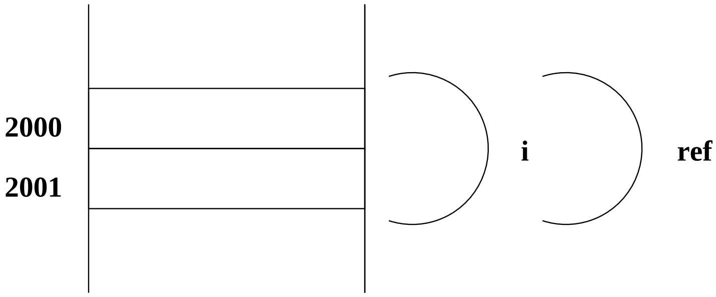
```
  ⋮

int i;
int &ref = i;

  ⋮
```

The following statements are equivalents ways to increase i :
`i++;` or `ref++;`

Both `i` and **ref** refer to the same memory location, for instance :



Functions( )

General form of a function :

```
Return-value FunctionName (arg1,arg2) {
     <body of function>
}
```

Example 1 : Function that computes the square of a number :

```
float SQUARE (float x) {
    return x*x;
}
```

Example 2 :

```
#include <iostream.h>
void WrongIncrease(int i) {
    i++;
}
void Increase(int &i) {
    i++;
}

int main() {
    int x=0;
    int y=0;

    WrongIncrease(x);
    Increase(y);

    cout << "x = " << x << "y = " << y << "\n";
    return 0 ;
}
```

Output of the program:

```
    x = 0 y = 1
```

Note : `void` return value indicates function returns no value.

- In function `WrongIncrease` argument is by value : value of `x` is copied into local variable `i`, `i` gets increased but `x` remains the same.
- In function `Increase` argument is by reference : reference of `y` in passed to local reference variable `i`, when `i` gets increased `y` being increased `i` becomes synonym to `y`).

Arrays [ ]

- Collection of n objects of given type, indexed from 0 to n-1.

Example :

```
   ⋮

int a[10];
int i;

for (i=0; i<10; i++) {
    a[i] = i + 3;
}

// print array in reverse order
for (i=9; i>=0; i--) {
    cout << "Pos " << i << "contains " << a[i] << "\n";
}

   ⋮
```

Result :

	0	1	2	3	4	5	6	7	8	9
a =	3	4	5	6	7	8	9	10	11	12

Output :

```
Pos 9 contains 12
Pos 8 contains 11
Pos 0 contains 3
```

Other examples of array declarations:

```
float v[3]; // array of 3 floats;
            // v[0], v[1], v[2]

int a[2][5];  // two arrays of five int's
```

```
char* vpc[32]; // arrays of 32 pointers to chair
```

Examples of array initialization :

```
int a[] = {3,4,5,6,7,8,9,10,11,12};
```

No need to specify size, the compiler assigns 10 positions automatically.

```
char vowels[] = "aeiou"; // 5 positions
```

Note that the following is invalid :

```
char vowels[5];
vowels = "aeiou"; // error
```

Because assignment is not defined for array type.