Structure Of C++ Program

The following example illustrate the structure of a C++ program:

• //Program 1.1
  //hello.cc(program name)
  //This program displays the message 'Hello World'
  #include<iostream.h>
  void main()
  {
    cout<<"Hello World"; // The message 'Hello World' is displayed
  }

  This Program displays the message 'Hello World' on the screen.

• The comment entries start with a // symbol and terminate at the end of the line. In Program 1.1, the lines. Multiple line comment entries can be included in a C++ program using the /* and the */ symbols.
C++ Data Types

C++ data types fall into three categories:

A. Simple: Three categories of simple data type
   • Integral: integers (numbers without a decimal)
   • Floating-point: decimal numbers
   • Enumeration: user-defined data type

B. Structured: Abstract Data Structures

C. Pointers: Will be described in the course

The Integral data types are further classified into nine categories: – char, short, int, long, bool – unsigned char, unsigned short, unsigned int, unsigned long
Allocating Memory
There are Two Methods for allocating Memory:

A. Named constant: memory location whose content can’t change during execution:
   The syntax to declare a named constant is:

   ```
   const  Data-Type  Identifier  =  Value;
   ```

In C++, const is a reserved

B. Variable: memory location whose content may change during execution
   The syntax to declare a Variable constant is:

   ```
   Data-Type  Identifier,  Identifier,  ....;
   ```
Identifier, Variable, Declaration, and Assignment

Identifier consists of letters, digits, and underscores, and begins with letter or underscore.

A. The variable is an identifier and the syntax of declaration of variable is:

Data-type variable-name, variable name, ..;

B. The syntax of Assignment Statement is:

Variable-name = Expression;

C. The syntax of Declaration and Assignment Statement is:

Data-type variable-name = Expression;
Fundamental (Basic) Data Types

The fundamental data types are at the lowest level, which means that they are used for actual data representation in the memory. The fundamental data types supported by C++ are listed in Table 1.1.

Table 1.1

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Numbers of Bytes</th>
<th>Used for Representing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>char</strong></td>
<td>1</td>
<td>characters and strings</td>
</tr>
<tr>
<td><strong>bool</strong></td>
<td>1</td>
<td>used for Boolean values</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td>4</td>
<td>integers</td>
</tr>
<tr>
<td><strong>float</strong></td>
<td>4</td>
<td>floating point numbers, -3.4E+38 to 3.4E+38</td>
</tr>
<tr>
<td><strong>double</strong></td>
<td>8</td>
<td>any real number, -1.7E+308 to 1.7E+308</td>
</tr>
</tbody>
</table>

Variables in C++

Variables are fundamental to any language. Values can be assigned to variable, which can be changed in the course of program execution. The value assigned to a variable is placed in the memory allocated to that variable. Variables can be created using the keywords `int`, `char`, `bool`, `float`, and `double`.

**Integer Variables**

Examples: 

```cpp
int iVar1; //Define an integer variable iVar
int iVar2 = 10, iVar3; //Define the integer variable iVar2 and assigns a value 10 // to it, and, also defines a variable iVar3
```
Character Variables

Example: char cVar1; //Define a character variable cVar1
        char cVar2 = 'A'; //Define a character variable and assign a character constant

Float Variables:

Examples: float fVar1 = 20.50; //Define a float variable and assigns a value to it float fVar2; //Define a float variable.

Boolean Variables:

Examples: bool fVar1 = 'T';

Defining Strings:

Examples: char cWord[10];
            char text[40]; //Allocates 40 bytes the variable text
Input and Output Using `cin` and `cout`

The following program illustrates the usage and application of `cin` and `cout`:

```cpp
//This program demonstrates the usage
#include<iostream.h>
void main()
{
    float fTemp;
    float fCels;
    cout<<"Enter the temperature in farenheit: ";
    cin>>fTemp;
    fCels = (fTemp - 32) * 5/9;
    cout<<"The equivalent temperature in celsius is: "<<fCels<<endl;
}
```

The statement:

```cpp
    cin>>fTemp;
```
if..else construct

The following is the syntax of if..else:

if (Conditions)  Compound-Statement
else Compound-Statement ;

Where Compound-Statement means zero or more Statements. if and else are reserved words.

Compound-Statement = [ statement-1; statement-2; 
statemanet-3; ..;statement-n ] ;

Where n>=0
The following program illustrates the usage of the if..else construct:

```cpp
//This program demonstrates the if..else construct
#include<iostream.h>
void main()
{
    int iVar1, iVar2;
    cout<<"Input the first number:"; 
    cin>>iVar1;
    cout<<"Input the second number:";
    cin>>iVar2;
    if(iVar1 == iVar2)
    {
        cout<<"iVar1 is equal to iVar2";
    }
    else
    {
        cout<<"iVar1 is not equal to iVar2";
    }
}
```

The Operators Used in C++

All computer languages provide tools for some predefined operators. These tools are known as operators. The various categories of operators are:

1. Arithmetic operators
2. Relational operators
3. Logical operators
1. Arithmetic Operators

In addition to the four arithmetic operators +, -, *, /, C++ also provides the remainder or the modulo operator represented by the % symbol. The modulo operator returns the remainder when an integer is divided by another.

Note: The modulo operator works only with integers

The following program illustrates the usage of the modulo operator.

```cpp
int main()
{
    int Var1 = 11, iVar2 = 5, iVar3;
    iVar3 = iVar1 % iVar2;
    cout<<"The remainder is:"<<iVar3<<endl;
}
```

The output of Program 1.9 is:
The remainder is: 1
C++ offers a condensed approach to perform calculations using arithmetic assignment operators which combine an arithmetic operator (+, -, /, %) and the assignment operator (=). The statement,

\[ i\text{Count} = i\text{Count} + 2; \]

increments the value of the variable iCount by 2. The above statement can be modified as,

\[ i\text{Count} += 2; \]

using the arithmetic assignment operator (+=).

Other arithmetic assignment operators are: -=, *~, /= and %=.
2. Relational Operators

Relational operators are used to compare two values. The result of the comparison is either true(1) or false(0). Table below lists the various relational Operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
</tbody>
</table>

3. Logical Operators

Logical operators are used to combine two or more test expressions. C++ provides the AND(&&), the OR(||) and the NOT(!) logical operators. The && logical operator combines two conditional expressions and evaluates to true only if both conditions are fulfilled. The following program illustrate the usage of the && logical operator.

Continue Logical Operators

//This program demonstrate the usage of the && operator
void main()
{
    int iTemp1;
    cout<<"Enter a Number:"<<endl;
    cin>>iTemp1;
    if((iTemp1 != 0) && ((iTemp1 % 2) == 0))
    {
        cout<<"Even number";
    }
    else
    {
        cout<<"Odd number or number is zero";
    }
}

only if both the condition are true. Otherwise, the statement following else is executed.

Continue Logical Operators

The \( || \) logical operator combines two conditional expressions, and evaluates to true if any one condition is fulfilled, as seen below.

```cpp
//This program demonstrates the usage of the \( || \) logical operator.
int main()
{
    char cVar;
    cout<<"Enter A or a:";
    cin>>cVar;
    if(cVar == 'a' || cVar == 'A')
    {
        cout<<"You have entered the right character";
    }
    else
    {
        cout<<"The character is not A or a";
    }
}
```
Unary Operators

Unary operators operate on one operand (variable or constant). There are two types of unary operators - increment and decrement.

Increment Operator (++) and decrement Operator (--) are a unary operator, which increments, or decrement the value of a variable or constant by one. For example, to increment the value of a variable by one, the following statement can be used:

iVar = iVar + 1;

The above statement can also be written using an increment operator in the following way:

++iVar;

The increment and decrement operators can be used in two ways - prefix and postfix.

As a prefix, the operator precedes the variable. For examples:

++iVar; --iVar;

As a postfix, the operator follows the variable. For examples:

iVar++; iVar--;
Applications of Unary Operators

The following code segment differentiates the two notations for increment operation:

iVar1 = 10;
iVar2 = ++iVar1;
the equivalent of this code is:
iVar1 = 10;
iVar1 = iVar1 + 1;
iVar2 = iVar1;
In this case, both iVar1 and iVar2 are set to 11 because, in the prefix notation, the increment operation is performed prior to assignment.

However, if the code is written as:
iVar1 = 10;
iVar2 = iVar1++; 
the equivalent code will be:
iVar1 = 10;
iVar2 = iVar1;
iVar1 = iVar1 + 1;
Here, iVar1 is set to 11 but the value of iVar2 is 10 because, in the postfix notation.

C++ has a built-in multiple branch selection statement called switch. The switch statement successively tests the value of an expression against a list of integer or character constants. When a match is found, the statements associated with that constant are executed. The value-1, value-2, ..,value-n are the values of the variable-name during the test. The Syntax of (switch .. Case) is:

```
Switch (variable-name) 
{
    Case value-1: 
        {c++ statements;  break;}
    Case value-2: 
        {c++ statements;  break;}
    Case value-n: 
        {c++ statements; break;}
    default: 
        {c++ statements; break;}
}
```
Consider the following if..else statements:

```c
if(iNum1 > iNum2)
{
    iMax = iNum1;
}
else
{
    iMax = iNum2;
}
```

In the above program code, iMax is assigned the value iNum1 if the test is true, and is assigned the value iNum2 if the test is false.

The above code can be modified using the conditional operator as:

```c
iMax = (iNum1 > iNum2) ? iNum1 : iNum2;
```

The `?` operator is called the conditional (or the ternary) operator and the syntax take the form, `v = Exp1 ? Exp2 : Exp3;`

Where `Exp1`, `Exp2` and `Exp3` are expressions. `Exp1` is evaluated. If it is true, the value of `Exp2` is assigned to `v`, and if is false, then the value of `Exp3` is assigned to `v`.
Loop Construct

A loop causes a section of the program to be repeated a certain number of times. The repetition continues while a condition is true. When the condition becomes false, the loop ends and the control is passed to the statement following the loop.

There are three Loop Construct, these are:

1. While Loop
2. Do while Loop
3. For Loop
1. While Loop

The while construct repeats the body of the loop as long as the loop conditions are true.

The syntax of while loop is:
```c++
While (conditions)
{
  Body of the loop (Statement-1;statement-2;..;statement-n;)
}
```

Where statement-1, statement-2,.., statement-n are c++ statements and n>=0
Application of while Loop

//This program calculates the square of integer numbers
void main()
{
    char cReply;
    int iNum, iSquare;
    cout<<"Do you want to find the square of a number (y/n)?"; 
    cin>>cReply;
    while(cReply == 'y')
    {
        cout<<"Enter a number:'"; 
        cin>>iNum;
        iSquare = iNum * iNum;
        cout<<"The square of "<<iNum<<"is"<<iSquare<<endl;
        cout<<"Do you want to find the square of another number (y/n)?"; 
        cin>>cReply;
    }
}
2. do..while loop

In a while loop, the test expression is evaluated at the beginning of the loop. If the test expression is false, the loop body is not executed at all. If the loop body must be executed at least once, then the do..while construct should be used. The do..while construct places the test expression at the end of the loop. The syntax do..while is:

do
{
Body of the loop (Statement-1;statement-2;..;statement-n;)
}
while (conditions);
3. \textit{for} loop

The while and the do..while loops are used when the number of iteration (the number of times the loop body is executed) is not known. The \textit{for} loop is used for fixed iterations. The syntax of \textit{for} loop is:

\begin{verbatim}
for (Initialization Expression; Test Expression; Increment/Decrement Expression)
{
Body of the loop (Statement-1;statement-2;..;statement-n;)
}
\end{verbatim}
Program using break statement

```cpp
int main()
{
    float fNum;
    char cReply;
    do
    {
        cout<<"Enter a number:";
        cin>>fNum;
        if(fNum == 0)
            break;
        cout<<"Inverse of the number is "<<1/fNum<<endl;
        cout<<"Do you want to input another number(y/n)?";
        cin>>cReply;
    } while(cReply != 'n');
}
```

Program using continue statement:

    void main()
    {
        int iNum;
        char cReply = 'y';
        do
        {
            cout<<"Enter a number:";
            cin>>iNum;
            if(iNum > 100)
            {
                cout<<"The number is greater than 100, enter another"<<endl;
                continue;
            }
            cout<<"Do you want to enter another(y/n)?";
            cin>>cReply;
        } while(cReply != 'n');
    }
One and two Dimensional Arrays

An array is a collection of elements of the same data type are reference by a common name. Each element of an array can be referred to by an array name and a subscript or index. Arrays can be one-dimensional or two-dimensional.

The syntax of one-dimensional array is:

```
data-type var-name[size];
```

element:

```
char cArr[10];
```
Declaring One-dimensional Arrays

//This program performs input-output operations on arrays
void main()
{
    int iArr[8]; //Array of size 8 is defined
    int i,j;
    cout<<"Input the elements of the array:"<<endl;
    for(i=0; i<8; i++) //As last subscript is 7
    {
        cin>>iArr[i]; //i is called the array subscript
    }
    cout<<"The elements stored in the array are:"<<endl;
    for(j=0; j<8; j++)
    {
        cout<<iArr[j]<<""; //Displays the elements of the array
    }
}

Array iArr is declared to be of type int and size 8. A maximum of 8 numbers can be stored in the array and the individual elements are referenced using the array subscript.

How the elements are store in array

Assume after run the program the user enter the number: 1,2,7,9,13,16,3,8. Then elements stored in the array are:

<table>
<thead>
<tr>
<th></th>
<th>←---iArr[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>←---iArr[1]</td>
</tr>
<tr>
<td>7</td>
<td>←---iArr[2]</td>
</tr>
<tr>
<td>9</td>
<td>←---iArr[3]</td>
</tr>
<tr>
<td>13</td>
<td>←---iArr[4]</td>
</tr>
<tr>
<td>16</td>
<td>←---iArr[5]</td>
</tr>
<tr>
<td>3</td>
<td>←---iArr[6]</td>
</tr>
<tr>
<td>8</td>
<td>←---iArr[7]</td>
</tr>
</tbody>
</table>
Two Dimensional Arrays

typical two-dimensional array is like a time-table. To locate a piece of information, you determine the required row and column and then read the location where they meet. In the same way, a two dimensional array is a grid containing rows and columns in which each element is uniquely specified by means of its row and column coordinates. Two-dimensional character arrays hold an array of strings wherein a row represents a string and a column represents a single character in each of the strings.

The syntax of the general form of declaration of a two-dimensional array is:

```
data-type arrayname[x][y];
```

where 'x' is the number of rows and 'y' is the number of columns.

Example: a two-dimensional string array is:
```
char arrayname[x][y];
```
where 'x' is the number of rows and 'y' is the number of columns. If x=0 and y=10 means the two dimensional array is a string with 10 characters.

The string can be declared as one dimensional array as:
```
char arrayname[10];
```
For example, if there are seven strings in an array and the length of the longest string is nine, the array can be declared in the following manner:

```c
char cWeekdays[7][10];
```

**Initializing Two-dimensional Arrays**

The rules for initializing two-dimensional arrays are the same as for one-dimensional arrays. For example, to declare and initialize an array that would hold the days of the week, the array definition would be:

```c
char cWeekdays[7][10] = {
    "Sunday",
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday"
};
```
Introduction to Pointers

A pointer is a variable that holds an address. Every variable declared in a program has two components: 1. Address and 2. Value.

C++ has two unary operators for referring to the components of a variable. The first operator, '&', known as the address operator returns the address of the variable. The operator '& is followed by a variable name. The second operator, '*', known as the indirection operator returns the value stored in any address in memory. The operator * is followed by an address of the variable. example:

```c++
void fn()
{
    int i = 10; //Statement 1
    int *ivar; //Statement 2
    ivar = &i; //Statement 3 - ivar now points to i
}
```

ivar is a pointer. the declaration of ivar in Statement 2 is preceded by the operator '*'. Pointer variables are declared like normal variables except for the addition of the unary operator '*'. Since a pointer stores the address of another variable, notice that in Statement 3, ivar stores the address of the variable i.
Continue of Pointer

The example can be explained in a better way by referring to the following diagram.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Memory Content</th>
<th>Memory Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>10</td>
<td>0x102</td>
</tr>
<tr>
<td>iVar</td>
<td>0x102</td>
<td>0x106</td>
</tr>
</tbody>
</table>

In the above diagram, the variable i, which is stored at the location 0x102, stores the value 10. The pointer iVar, which is stored at the location 0x106, stores the address of the variable i. To access the value of a variable through a pointer, indirection operator '*' is used.
Manipulation of Pointers

//Pointer manipulation through indirection operator
#include<iostream.h>
void main()
{
    int *iPtr, iVar = 10;
iPtr = &iVar; //Initializing the pointer with the address of iVar
cout<<"Value of iVar is "<<iVar<<endl;
    //The following statement also prints the value of iVar but through a pointer
cout<<"Value of iVar accessed through a pointer is 
"<<*iPtr<<endl;
}

The output of above is:
Value of iVar is 10
Value of iVar accessed through a pointer is 10