C++ SYNTAX

Variable and Constant Declaration

A) Declaring Variables:

DataType VariableName**;**

1) *DataType* may be any predefined C++ data type or a user defined data type.

2) Some predefined C++ data types are *int, float,* and *char*.

3) Terminate declaration with a semicolon, **;**

4) Variables must be declared before they can be used but may be declared anywhere in program.

5) Constraints on naming variables and constants;
   a) must begin with letter or underscore.
   b) next characters may be letter, underscore, or number.
   c) normally not more then 32 characters.
   d) C++ is case sensitive, and can not use C++ reserved words as variable names.

B) Variables may be initialized when declared.

DataType VariableName = ***Value;***

C) Constants **must** be initialized when declared.

const DataType ConstantName = ***Value;***

1) *const* is a reserved word must be all lower case.
2) If *DataType* is not given then by default it is *int*.

Note
- Customary to declare variables and constants at the beginning of the function.
- Normally first letter of variable is uppercase.
- Constants are normally all uppercase letters.

C++ EXAMPLES

A) Declaring some simple variables:

int Number;

float Average;

char Ch;

Declaring multiple variables of same data type, separate variable names with a comma.

int X, Y, Z;

float Time, Distance;

char Ch1, Answer;

B) Initialized variables:

int Sum = 0, Count = 0, Max = 100;

C) Constants:

const int LARGEST = 125**;**

const float Weight = 35.958**;**

C++ SYNTAX

Main function, Include, Input/Output

A) Basic C++ Program

```
#include <FileName>
void main()
{
   Variable Declarations;

   Statement1;
   Statement2;
      :
   StatementN;
}
```

1) All C++ programs begin execution in the function named *main*.

2) The pound sign, **#**, is used to signify a preprocessor directive. Used with *include* to tell preprocessor to actually include these other files when program is compiled.

3) FileName may be an already existing system file, ie, *iostream.h* or *conio.h*, or a user created file.

4) Function body starts with **{** and finishes with **}**.

5) The main function may be a void returning function or a value returning function. If value returning then must have *return **data**;* somewhere in function body. (see example to right)

B) Standard Input and Output:

Output:

```
cout << data;
```

cout stands for console output, << is the insertion operator and data may be either a variable, a constant or an expression.

Input:

```
cin >> VariableName;
```

cin stands for console input, >> is the extraction operator and VariableName is a declared variable.

C++ EXAMPLES

A) Simple void returning main function

```
#include <iostream.h>
void main()
{
  int X, Y, Sum;

  cout << "Enter two integers --> ";
  cin >> X;
  cin >> Y;
  Sum = X + Y;
  cout << "\nSum is " << Sum;
 }
```

Value returning main function

```
#include <iostream.h>
#include <conio.h>
int main()
{
  float N1 = 2.5, N2 = 4.5;

  clrscr();              //clears output screen
  cout << "N1 divided by N2 is " << N1/N2;
  cout << "N2 divided by N1 is " << N2/N1;
  return 0;
}
```

C++ SYNTAX

IF Statements

A) Basic IF statement with single statement:

```
if (expression)
    statement;
```

When the expression has a value of nonzero, then this is considered true and the statement associated with the *if* statement is executed, otherwise an expression value of zero is considered false and statement is ignored.

B) Basic IF statement with compound statements:

```
if (expression)
  {
    statement1;
    statement2;
        :
    statementN;
  }
```

C) IF- ELSE statement

```
if (expression)
   statement1;
else
   statement2;
```

D) IF- ELSE- IF statement

```
if (expression1)
   statement1;
else if (expression2)
   statement2;
else if (expression3)
   statement3;
else
   statement4;
```

Note any statement could actually be a compound statement. A compound statement starts with **{** and ends with **}**.

Also in if-else-if, the last *else* is optional.

C++ EXAMPLES

A) Simple If statement

```
if ( X >= 25)
   cout << "This is good";
```

```
if (Ch >= 'A' && Ch <= 'Z')
   cout << "This is a capital letter";
```

B) Compound If

```
if ( N == 0)
  {
    cout << "N is zero";
    cout << "Please enter next number ";
    cin >> X;
  }
```

C) If - else example

```
if (X > 0)
   cout << "Positive";
else
   cout << "Negative";
```

D) If - else - if

```
if (Y == 0)
   N = 1;
else if (Y == 1)
   N = N;
else if (Y == 2)
   N = N * N;
else
   N = N * N * N;
```

C++ SYNTAX

While, Do-While, and For loops

A) The while loop statement:

```
while (expression)
 {
   statement1;
   statement2;
       :
   statementN;
 }
```

B) The do-while loop statement:

```
do
  {
    statement1;
    statement2;
        :
    statementN;
  }
while (expression);
```

C) The for loop statement:

```
                Exp1            Exp2      Exp3
for ( LCV = initial value; expression; change LCV )
  {
    statement1;
    statement2;
        :
    statementN;
  }
```

Note Always make sure you do following:

1) Initialize LCV (Loop Control Variable)
2) Test LCV
3) Update/change LCV

All loops will loop when value of expression is true or a non-zero number. Value of zero, 0, corresponds to false expression, looping stops

C++ EXAMPLES

A) While example

```
int X = 0, Total = 0;
while ( X < 10 )
 {
   if (X%3 == 0)
       Total = Total + X;
   else
       cout << "Not divisible by 3";
    X++;
 }
```

B) Do-While example

```
do
  {
    cin.get(Ch);
    if (Ch >= 'A' && Ch <= 'Z')
        Capital++;
  }
while (Ch != '\n');
```

C) For example

```
for (int k = 1; k <= 5; k++)
  {
    cout << "Enter number --> ";
    cin >> Num;
    Total = Total + Num;
  }
```

C++ SYNTAX

Functions

In C++ there are no procedures, everything is called a function; even main is actually a special function that is called main. This is where all programs begin execution, the main function can then call other functions; either user defined or library functions found in header files.

The general FUNCTION RULE:
Functions should be designed to perform a single, well-defined task which is logically coherent. The Function name should tell what that task is.

SYNTAX - (Declaring a function)

```
ReturnValueDataType  FunctionName ( formal parameter list )
{
      local variable declarations;
      C++ statements;
}
```

SYNTAX - (Calling function) (Two possible ways or usage)

FunctionName (*actual parameter list* **);** // call for a *void* returning function
VariableName = FunctionName (*actual parameter list* **);** // function returns data

Comments

1) All functions must be declared or prototyped before the main function.

2) ReturnValueDataType tells compiler data type of returned data passed back via the function name. By default it is type *int*, if want to return nothing then use *void*.

3) Formal parameter list tells compiler:
   a) variable names used in parameter list
   b) data type associated with each variable in list
   c) the variables location in list
   d) what kind of parameters are they, i.e. Value or Reference Parameters
   e) variables in formal parameter list are also considered local variables to function
   f) no formal parameters then use the word *void* inside the parentheses

4) Value Parameters
   a) used as input to function
   b) passes data from main (or other functions) into declared function

5) Reference Parameters
   a) normally used as output from the function, but can also be used for input
   b) passes data from declared function back to function called from
   c) each reference parameter must have &, ampersand, before it in parameter list.

6) Correspondence Rules between formal and actual parameters (generally true)

a) Actual parameter list must be the same size and order as the formal list.
b) Actual parameter that corresponds to a VALUE formal parameter:
    -) may be a data type compatible to formal parameter
    -) may be a variable, expression, or constant
c) Actual parameter that corresponds to a REFERENCE formal parameter
    -) should be of the same data type as the formal parameter
    -) must be a variable itself, not an expression or constant

7) When function is declared as type *void*, then calling function is a C++ statement in itself and function name cannot be used inside an expression.

FUNCTION PROTOTYPES

SYNTAX

ReturnValueDataType **FunctionName** (*data types of formal parameters*) **;**

1) Prototypes are placed before the main function
2) Function prototypes tell compiler
    a) data type of returned value of function, if any, i.e. could be *void*
    b) number of parameters
    c) data type of each parameter
    d) order of parameters in list
    e) if parameter is value or reference parameter
3) Variable name not needed in function prototype
4) Must have the semicolon at the end
5) After the main function then you have the formal function declaration, as described above.