Basic of C++

- Character set of c++ :-

A to Z } alphabets
a to z }
0 to 9 digits
+, -, {, <, ..., } special symbols.

c++ is case sensitive and it differentiate between upper case & lower case alphabate

* Tokens :-

The smallest individual unit in a program are known as tokens.
c++ has following tokens such as

1) keywords
2) identifiers
3) constants
4) Strings
5) Operators.

* Keywords :-

The keyword implement specific c language feature. They are explicitly reserve identifiers and can not be used as name for the program variable or other user define program elements.

The table shows complete set of keywords most of them are similar to C with additional new one.

ASM, auto, break, case, catch, char, class, const, char, continue, default, delete, do, double, else, enun, extern, float, for, friend, go to, if, inline, int, long, new, operator, private, protected public, register, return, short, Signed, size of, static, struct, switch, template this, try, typedef, throw, union, unsigned, virtual, void, volatile, while

* Identifiers :-

Identifiers refered as name of the variable, function, array created by the programmer.

They are fundamental, requirement or any language.

Following are the rules for naming these identifiers.

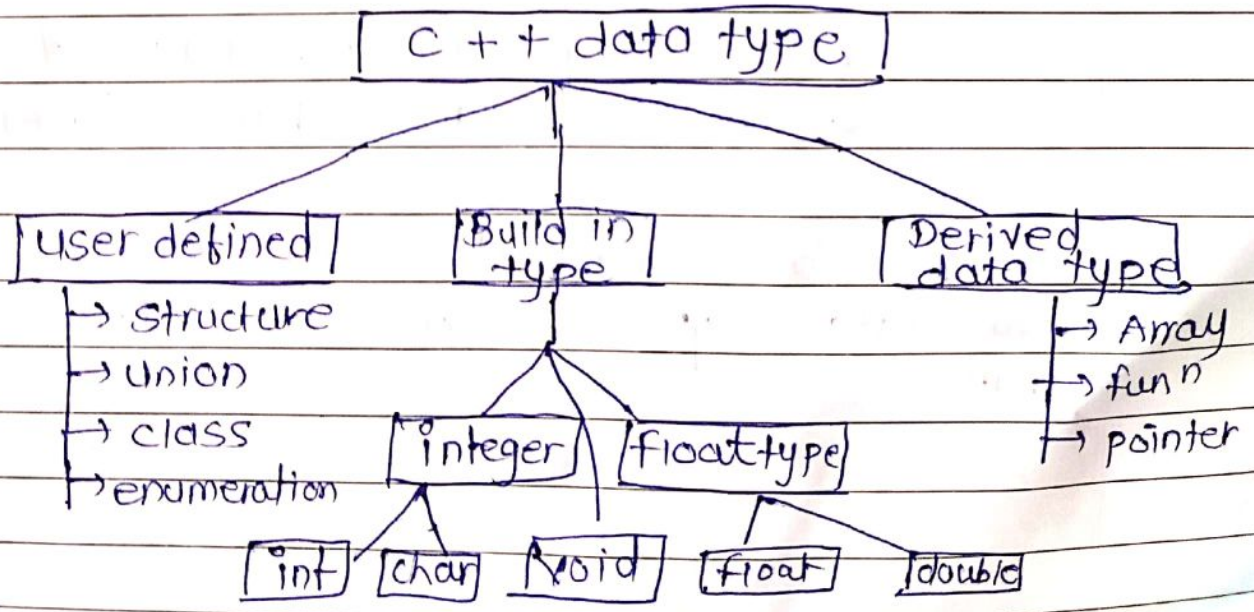
1) Only alphabetic characters, digits and are permitted.

2) The name cannot start with digits

3) Upper and lower case letters are distinct.

4) A declared keyword cannot be used as variable name.

* Basic data type :-



* User defined data type.

We already defined data type such as struct and union while these data types are legal in c++ but some more features have been added to make them suitable for OOP.

The class is one of the example of user defined data type and object is variable of that data type.

* Enumeration

Is another user defined data type which provide way for attaching names to number their by increasing compressebility of the code.

A ~~new key~~ enum keyword automatically enumate list of words by assigning the values 0, 1, 2, ...

e.g.

enum shape (circle, squar, triangle)
enum color (red, blue, green)

in above example shape & color are ~~new~~ data type & use the ~~data~~ circle, enum

Square and triangle with 0, 1, 2 number.

8/07/2019

* Derived data types

- Arrays :-

The application of arrays in c++ is similar to that in c. The only exception is the way character arrays are initialize.

In c++, the size should be one larger that the no. of characters in the string. This is special for character array.

- Functions :-

Functions have under gone major changes in c++. Many of these modifications and improvements where driven by requirement of object oriented concept in c++.

- Pointer :-

Pointers are declared and initialize as in c.

for example

```
int *ip; // integer pointer
ip = &x; // address of x is assigned to ip
*ip = 50; // 50 is assigned to x through direction
```

Pointers are expensively use in c++ for & memory management and achieving polymorphism.

* Symbolic Constants :-

There are two ways of creating symbolic constants in c++.

i) Using the qualifier const
ii) Defining a set of integer constant using enum keyword.

for example,

```
const int size = 10;
char name[size];
```

The name constant set the variable value to the value assign to it, and cannot be change throughout the program.

Another method of naming the integer constant is as follows.

ex:-

```
enum {x, y, z};
```

This defines x, y and z as integer constant with the values 0, 1, 2 respectively

This is equivalent to

```
const x = 0;
const y = 1;
const z = 2;
```

We can also assign values to x, y, z explicitly.

```
enum { x = 50, y = 100, z = 200 }
```

Such values can be any integer values

* Declaration of Variables

C++ allows the declaration of variable anywhere in the scope. This means a variable can be declared at the place of its first use.

This makes the program much easier to write and reduces the error, and makes the program easier to understand.

```
Void main ()
{
int = m1, m2, total;
m1 = 50;
m2 = 115;
total = m1 + m2;
float avg = total/2;
Cont << average = " << avg;
}
```

e.g.

```
for (int i=1; i<=5; i++);
```

* Dynamic initialization of Variable :-

One additional feature of c++ is that it permits initialization of variable at run time. This is refered to as dynamic initialization.

In c++, a variable can be initialize at one tu time using expression at the place of declaration.

For example :-

```
float area = 3.14*r*r;
```

or

```
float avg = total/i;
```

* Reference Variable :-

c++ introduces a new kind of variable known as the reference variable.

A reference variable provides and alais name (alternative name) for previously defined variable.

For. eg

If we make the variable 'sum' a reference to the variable 'total' then sum and total can be used interchangebly to represent that variable.

A reference variable is created as follows,

data type &reference name = Variable name

e.g.

```
float total = 100;
float &sum  = total;
```

total is a float type variable which ha been already declared, sum is the alternative name declared to represent the variable total. Both the variables refer to the same data object in the memory.

Now the statement,

```
cout<< total;
cout<< sum;
```

both the statement print the value 100

OR

```
if total = total + 10;
    cout<< sum;
```

it will print the sum value 110.

* Operators in c++

All c operators are valid in c++ in addition to that c++ introduces to ~~that c++ int~~ some new operators such as

<< → insertion operator

>> → Extraction operator

:: → Scope resolution operator

→* → Pointer to member operator

.* → Pointer to member operator

::* → Pointer to member declaratio

delete → Memory release operator

endl → Line feed operator

new → Memory allocation operator

setw → field build operator.

In addition, c++ allows to provide new definations to some of the building operators.

i.e We can give several meanings to an operator, this process is known as operator overloading.

- Scope resolution Operator (::)

c++ is block structured language, We know that the same variable ~~can~~ name can be used to have different meaning in different block.

A variable declared insert the block is said to be local to the block.

The variable declared out of any block is called global variable.

```
int x = 100;  // global
main()
  {
    {
      int x = 10 ;  // local to block 1
      cout << x ;  // it print x = 10
      cout << ::x ;  // it print x = 100
    }
    cout << ::x ;  // it print x = 100
    {
      float x = 10.10 ; // local to block 2
      cout << x ;  // it print x = 10.10
    }
  }
```

The new operator called the scope resolution operator can be used to uncover the hiddle variable, it takes the following form.

```
:: variable name
```

This operator allows access to the global version variable.

Following program illustrates this

```
// use of :: operator
#include <iostream.h>
int m = 10 ; // global
void main ()
{
  int m = 20; // m declared as an local to
  {                main ()
    int k = m;
    int m = 30 ; // local to inner block
    cout << " We are in inner block " \n";
    cout << " k = " << k << " \n " ;
    cout << " m = " << m << " \n " ;
    cout << " ::m = " << ::m << " \n " ;
  }
  cout << " m = " << m << " \n " ;
  cout << " ::m = " << ::m ;
}
```

output :-

We are in inner block
k = 20
m = 30
∴ m = 10
n = 20
∴ n = 10

* Memory Management Operator

Object can be created by using new and destroyed by using delete when required.

Data object created inside the block with new will remain in existance until it is explicitly destroyed by using delete.

Following is the general form:

Pointer variable = new datatype;

e.g.

```
p = new int;
q = new float;
```

p is a pointer of type int and q is a pointer of type float, remember that p and q must have already been declared as pointer of appropriate type.

You can specify values to the variable as follows.

```
int *p = new int (25);
float *q = new float (95.75);
```

Using delete operator we can delete the object which can no longer needed.

its general format is as follows

```
delete pointer variable ;
```

e.g.

```
delete p;
delete q;
```

2/07/2019

* Manipulator

Manipulators are operators that are used to format the data display. The most commonly used manipulators are endl and setw.

The endl manipulator when using out put statement causes a line feed to be inserted. It has the same effect as using the new line character (\n)

for example,

```
cout << " m = " << m << endl;
cout << " n = " << n << endl;
cout << " p = " << p; << endl;
```

This will display the output as

m = 100
n = 200
p = 300

if m, n and p are the value 100, 200 and 300 respectively.

The setw manipulator manipulate the width of numbers and force them to be printed ~~write~~ right justified.

For example :-

```
cout << " m = " << m ;
cout << " m = " << setw(5) << m;
cout << " m = " << setw(5) << 90;
```

output :-

m = [| | | | |]

m = [| | 1 | 0 | 0]

n = [| | 9 | 0]

```
// use of setw & endl operator.
# include <iostream.h>
# include <iomanip.h>  // for setw
main()
  {
   int Basic = 950, allowance = 95, total = 1045;
   cout << setw (10) << "Basic" << setw (10)
                << Basic << endl;
   cout << setw(10) << "allownce" << setw(10)
                << allownce << endl;
   cout << setw(10) << "total" << setw(10)
                << total << endl;
  }
```

output :-

```
     Basic     950
allownce        95
   .total      1045
```

* Typecast Operator :-

c++ permits explicit type conversion of variable or expression using type cast operator.

following is the way

type name (expression)

e.g. average = sum / float(i).

A type name behaves as it is a function for converting values to a designated type. In above example, the float type is applied to sum variable.

* Control Structures :-

In C++, a large number of functions are used that pass messages and process the data contained in objects.

A function is setup to perform a task, when the task is complex many different algorithms can be design to acheive the goal.

For solving any problem, every program uses any combination of following three control structures.

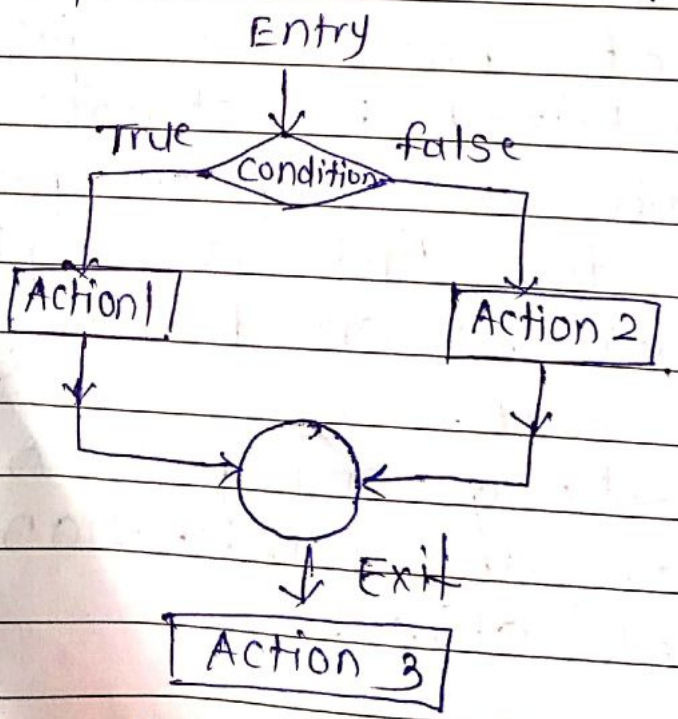
① Sequence structure (straight line)

② Selection structure (Branching)

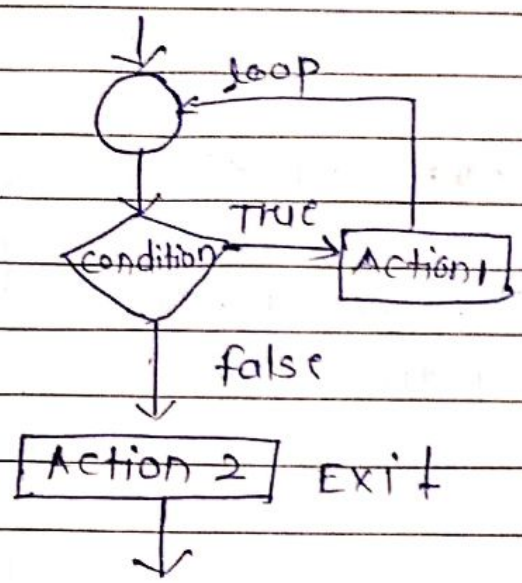
③ loop structure (itration or repetation)

The following flow diagram shows these all three type of structures.

a) Sequence Structure



b) Selection



c) loop / repetation structure.

* The if - statement :-

To perform the branching or selection structure the if - statement is implemented in two form.

① Simple if statement
② if - else statement

form - 1

```
if (expr true)
  {
   action 1;
  }
action 2;
action 3;
```

} simple if

form 2 :

```
if (expr true)
  {
   action 1;
   action 2;
  }
else
  {
   action 3;
   action 4;
```

if-else

Switch statement :-

It is used for multiple choice.

```
Switch (expression)
{
 case 1 :
   {action 1;
    break ;
 case 2 :
   {action 2;
    break ;
   }
 case 3 :
   {
   action 3;
   break ;
   }
  ;
  :
```

```
:
default
  {
    ...
    ...
  }
}
```

12/7/19.

* Loop Structure :-

- While Statement

This is a loop structure but, it is an entry control loop means it first check the condition and if the condition is true then only it enters into the loop.

For every loop execution three things are necessary.

i) Initialization of loop counter
ii) Loop condition or test condition.
iii) Increment or decrement of loop counter variable.

The Syntax of while loop is

```
While (conditime)
  {
    do action 1;
    ---
    ---
    increment / decrement
  }
    do action 2;
```

do-while statement :

do-while is an exit control loop means action will be perform first then the condition is tested for true or false.

In this type of structure the loop statements are executed at least once even though the condition is false at first time.

The Syntax is

```
do
  {
  action 1;
  ____
  ____
  ____
  }
  while (condition = true);
```

The ~~for~~ for statement :-

The for is also entry control loop, and is use when an action is to be repeated for a predetermine no. of times.

The syntax is

```
for( initial value; test condition; incr/decr)
  {
    ------
    ------
    ------
  }
```

Write a program to print the following output using for loop.

```
* * * * *
* * * *
* * *
* *
*
```