C++ SHORT HANDWRITTEN NOTES WITH OOP'S CONCEPTS

Support me on | Copyrighted by :

Instagram : @curious..programmer
TELEGRAM : @curious-coder
YOUTUBE : @yadnyesh patil

HAPPY LEARNING!

chapter (1)

Beginning with c++

Date / /
Page 1 L&W

Introduction :-

c++ is an object oriented programing language. It is an extension of c programming language.

c++ is a powerful general purpose programming language. It can be used to develop operating system, browsers, games and so on.

Types of programming

(A) Procedure - oriented programming
(B) Object oriented programming

(A) Procedure - oriented programming :-

Conventional programming, using high level languages such as COBOL, FORTRAN and C is commonly known as procedure - oriented programming (pop).

In procedure - oriented programming the problem is viewed as a sequence of thing to be executed such as reading, calculating and printing. A number of functions are written to accomplish these tasks. The primary focus is on functions.

(B) Object oriented programming :-

Object oriented programming is a methadology to design a program with the help of objects and classes. The major motivating factor in the invention of object-oriented approach is to remove some of the flaws in pop approach.

Basic Concepts of Objects - oriented programming

- Objects
- classes
- Data abstraction and Encapsulation
- Inheritance
- polymorphism
- Dynamic binding
- Message passing

Application of OOP

- Real time system
- Simulation and modeling
- object oriented database
- Hypertext, hypermedia, and expertext
- AI and expert systems
- Decision support and office automation systems
- CIM/ CAM/ CAD systems

Structure of c++ program :

c++ program structure is divided into various sections namely, headers, class defination, member functions definitions and main function.

c++ Headers

class definition

member functions definition

Main function

structure of c++ program

A sample c++ program (without class)

```
#include <iostream>
using namespace std;                    @curious..programmer
    int main()
    {
        cout << " first c++ program " ;
        return 0 ;
    }
```

Comments :

C++ introduces a new comment symbol ⟶ // (double slash).

Example : // This is Example of C++ Comment.

* Comments start with a double slash symbol and terminate at the end of the line.
* The C comment symbols /*, */ are still valid and more suitable for multiline comments.

```
/* This is an Example of multi-line
   C++ comment. */
```


Output operator :-

```
Cout << " C++ is better than C ";
```

The operator << is called the insertion or put operator

When one operator can be used for different purposes depending upon the context this concept is known as operator overloading.

Namespace :- This defines a scope for the identifiers that are used in a program.

Input operator :

```
cin >> number1 ;
```

The operator >> is known as extraction or get from operator.

It extracts / Takes the value from the keyboard and assigns it to the variable on its right.

Cascading of I/O operators :

```
cout << " sum = " << sum << "\n" ;
```

The multiple use of << (Insertion operator) in one statement is called cascading.

chapter ②

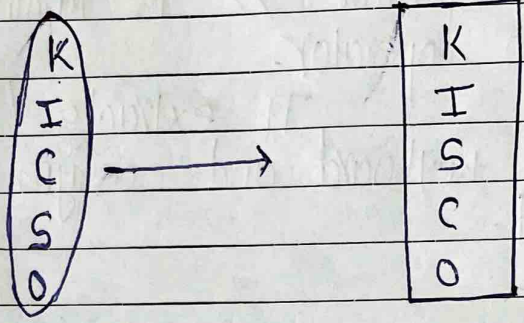
Tokens in C++

Date / /
Page 6 L&W

Tokens in C++

The smallest individual units in a program are known as tokens. C++ has the following tokens

- keywords
- Identifiers
- Constants
- Strings
- operators



(K) ⟶ Keywords

C++ Keywords

auto	double	long	switch
break	else	new	this
case	enum	public	throw
char	extern	private	try
class	float	protected	catch
const	flor	return	void
continue	friend	short	while
default	goto	sizeof	final
delete	if	static	union
do	int	struct	virtual

@curious_programmer

~~Input operato~~

Identifiers ⟶ (I)

Identifiers refer to the names of variables, functions arrays, classes etc, created by the programmer.

The following rules are common to both C and c++

- Only alphabetic characters, digits and underscores are permitted
- The name cannot start with digit
- Uppercase and lowercase letters are distinct

Constant ⟶ (C)

Whenever we declared any variable as a constant and assigning a particular value to it so we can't change that value further in a program

Example `Const int size = 10 ;`

chapter ③

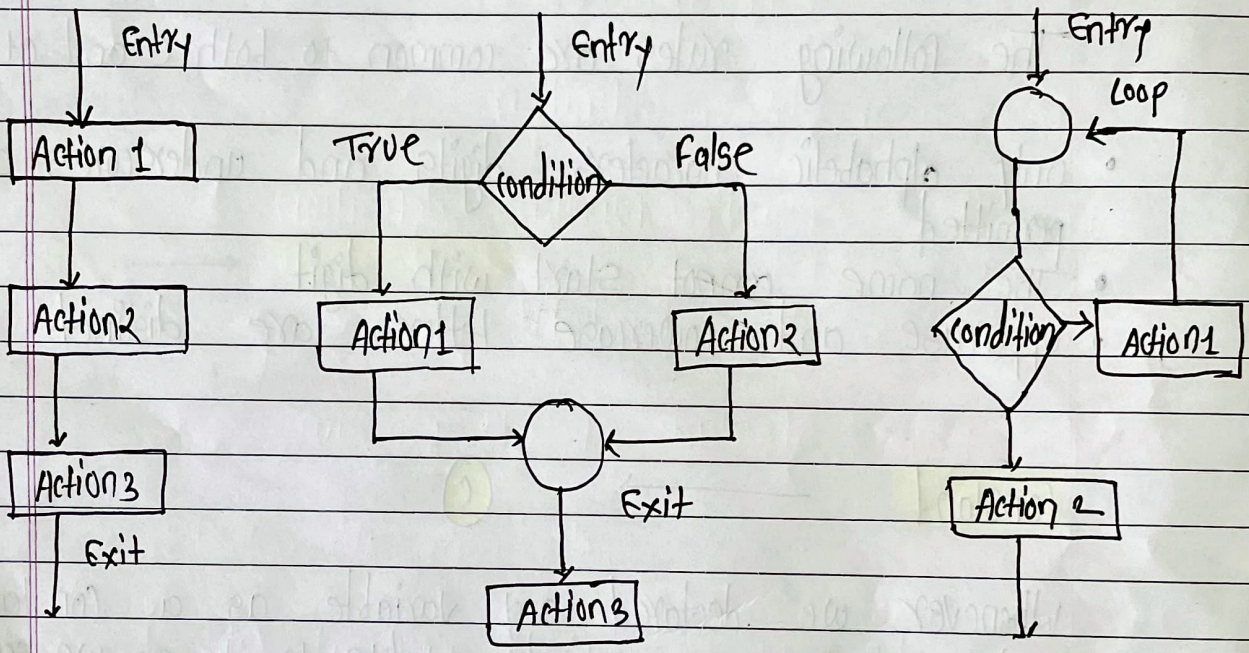
Control Structures

Date / /
Page 8 L&W

Control Structures :

Following are three control structures :

1. Sequence structure (straight lines)
2. Selection structure (Branching)
3. Loop structure (Iteration or repetition)



(a) Sequence (b) Selection (c) Loop

The if Statement :

The if statement is implemented in two forms

- simple if statement
- If.... else statement

@curious.programmer

Examples :

Form 1

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


Form 2

```
if ( expression is true)
  {
     action 1  ;
  }
else {
     action 2 ;
   }
     action 3  ;
```

The switch statement

This is multiple-branching statement where, based on a condition, the control is transferred to one of the many possible points. This is implemented as follows:

```
switch (Expression)
{
    case 1 ;  statement ;
              break ;
    case 2 :  statement ;
              break ;
       :
    case n :  statement ;
              break ;
  Default :   statement ;
              break ;
}
```

The do-while statement

The do-while statement is an exit-controlled loop. Based on a condition, the control is transferred back to a particular point in the program. The syntax is as follow:

```
do {
    action ;
   }
while (condition is true);
action 2 ;
```

The while statement

This is also a loop structure, but is an entry-controlled one. The syntax is as follows:

```
while ( condition is true)
  {
    action 1 ;
  }
  action 2 ;
```


The for statement :

The for is an entry-controlled loop and is used when an action is to be repeated for a predetermined number of times. The syntax is as follows

```
for( initial value ; condition ; increment/decrement)
   {
     action 1 ;
   }
   action 2 ;
```

chapter ④ Functions in C++

Functions Introduction :

We know that functions play an important role in C program development. functions are the set of program or set which is used to perform particular task. So it is possible to reduce the size of a program by calling and using them at different places in the program

```
void show() ;  /* function declaration
main()
{
  -----
  show() ;     /* function call
  -----
}
  void show()
  {
    -------
    -------
  }
```


Function prototyping :

Function prototype is a declaration statement in the calling program and is of following form

```
Type Function-Name ( argument - list ) ;
```

Example :

```
float volume (int x, float y, float z) ;
```

Call by Reference :

provision of the reference variable in c++ permits us to pass parameters to the functions by reference.

```
void add (int &a, int &b) // a and b are
{                            reference variables
    int t;
    t = a+b;
}
```

Now if a and b are two integer variables then the function call

```
add (a, b);
```

This function can be called as follows :

```
add (&a, &b);
```

This approach is called as call by refference method

Inline functions :

To eliminate the cost of small functions, c++ proposes a new feature called inline function. An inline function is a function that is expanded in a line when it is invoked.

```
inline function-header
{
    function body
}
```

@curious-programmer

Example :

```
inline double cube (double a)
{
    return (a*a*a);
}
```

Function Overloading :-

overloading refers to the use the same thing for different purposes. This means that we can use the same function name to create function that perform a variety of different tasks. This is known as function polymorphism in oop

Example :

```
int add ( int a, int b )
{
    int c = a+b ;
}
int add (int a, int b, int c)
{
    int d= a+b+c;
}
```

@curious_programmer

Math Library Functions

The standard c++ supports many math functions that can be used for performing certain common used calculations.

Function	purposes
ceil(x)	Rounds x to the smallest integer not less than x ceil(8.1) = 9.0.
cos(x)	trigonometric cosine of x
exp(x)	Exponential function e^x
fabs(x)	Absolute value of x

chapter (5)

classes and objects

Specifying a class

A class is a way to bind the data and its associated functions together. It allows the data to be hidden, if necessary from external use.

The general form of a class declaration is :

```
class class_name
{
    private :
        variable declarations ;
        function declarations ;
    public :
        variable declarations ;
        function declarations ;
};
```


creating objects

objects is nothing but the blue print of the class or an instances of the class. The class variable is called as **objects**

Syntax : `class_name object_name = New class_name ();`

Example : `Ved vedant = new Ved();`
Here Ved is a class and vedant is an object.

Accessing class members

We can access the class function by using object-Name or with the help of class-name object

Example : ved.add();

Syntax : object-Name-function-Name (actual-arguments);

Static Data members

A data member of a class can be qualified as static. The properties of a static member variables are similar to that of a c static variable. A static variable has a certain special characteristics these are :

- It is initialized to zero when the first object is created.
- only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.

Static member functions

Like static member variables, we can also have static member function. A member function that is declared static has the following properties.

- A static function can have access to only other static members declared in the same class.
- A static member function can be called using the class name

@curious_programmer

chapter (6) Constructors and Destructors

Constructors :

A constructor is a special member function called the constructor which task is to initialize the objects of its class. It is special because its name is the same as the class name. The constructor is invoked whenever an object of its associated class is created.

A constructor is declared and defined as follows :

```
class integer
{
    int m, n ;
public :  int m, n ;
          integer (void) ;
          ::::::
};
integer :: integer (void)
{
    m=0 ;
    n=0 ;
}
```


The constructor functions have some special characteristics these are

- They should be declared in the public section
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and therefore and they cannot return values.
- They cannot be inherited, though a derived class call the base class constructor.
- Like c++ functions, they can have default arguments.
- We cannot refer to their addresses.

parameterized constructors

The constructors with any list of parameters that is called as parametrized constructors.

Example :

```
class integer
{
    int m, n ;
    public :
    integer (int x, int y) ;
        -----
        -----
};
integer :: integer (int x, int y)
{
    m = x ;
    n = y ;
}
```

copy constructor

we can copying the value of a object to the another object by using copy constructor.

Example :

```
class code {
        int id ;
        public :  code() { }
                  code(int a) { id=a }
};
int main()
{
    code A(100) ;
    code B(A) ;
    code C = A;
}
```

@curious.programmer

chapter (7)

Inheritance : Extending classes

M T W T F S S
Page No.:
Date: YOUVA

Inheritance :

Inheritance is a mechanism to derived a new class based on the old class. so that the object of a new class can access all the properties of the old class.

Defining derived classes

```
class derived-class-name : visibility-mode base-class
    {
        ----
        ----
        ----
    };
```

@curious_programmer

Example :

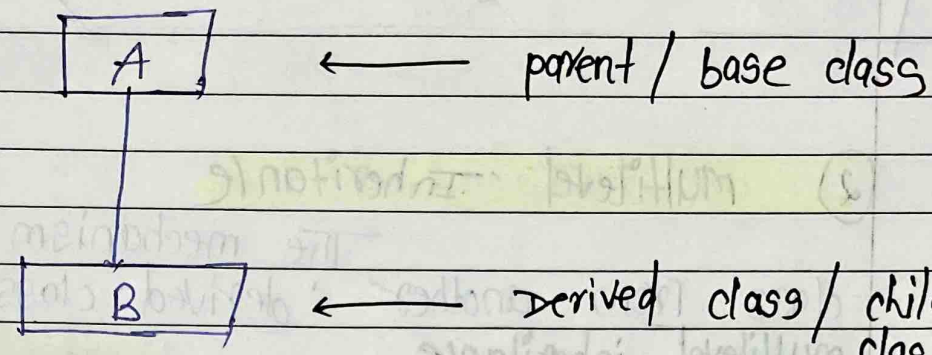
①
```
class derived : private base
    {
        member of derived // private derivation
    };
```

②
```
class derived : public base
    {
        member of derived // public
                             derivation
    };
```

Types of Inheritances

① Single Inheritance :-

When A derived class with only one base class is called as single inheritance



ⓐ Single Inheritance.

Example :

```
#include<iostream>
using namespace std;
class B {
    int a ;
public:
    int b ;
    void set_ab() ;
    int get_a(void) ;
    void show_a(void) ;
};
class B :: set_ab (void)
{
    int c ;
public:
    void mul(void) ;
    void display(void) ;
};
```

class D : public B

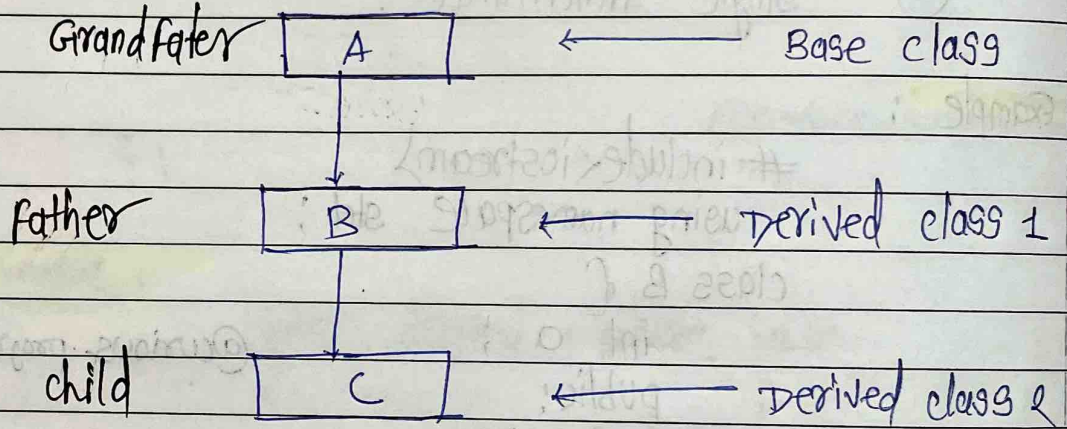
@curious_programmer

② Multilevel Inheritance

A derived class with several base classes is called as

② Multilevel Inheritance

The mechanism of deriving a class from another 'derived class' is known as multilevel inheritance



Example :

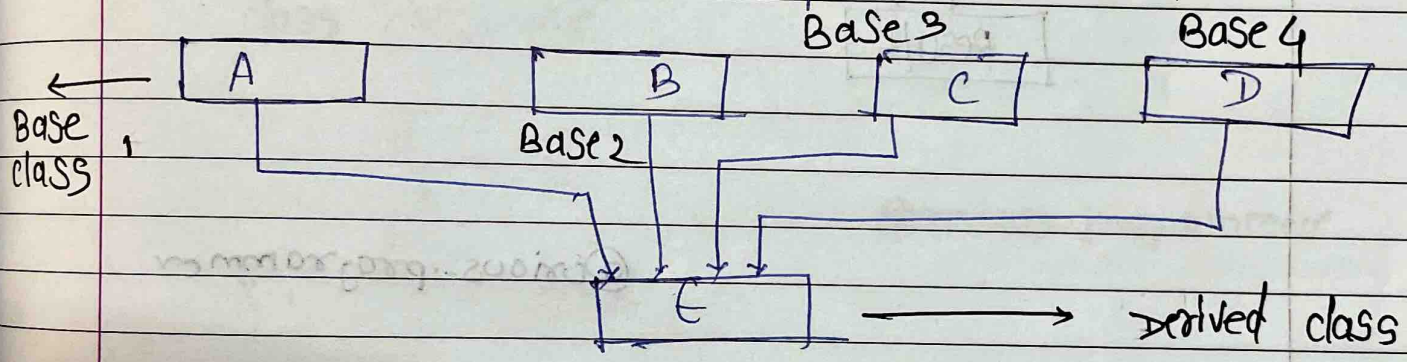
```
class student {
  protected :
    int roll_number ;
  public : void get_number(int) ;
           void put_number(void) ;
  } ;
void student :: get_number (int a)
{
  roll_number = a ;
}
void student :: put_number ()
{ cout << "Roll number : " << roll_number ;
}
```

```
class Test : public student
{
  }

class result : public test
   {
      ::::
     }
```

(3) **multiple Inheritance** :

A derived class with several base classes is called as multiple inheritance.



Example :

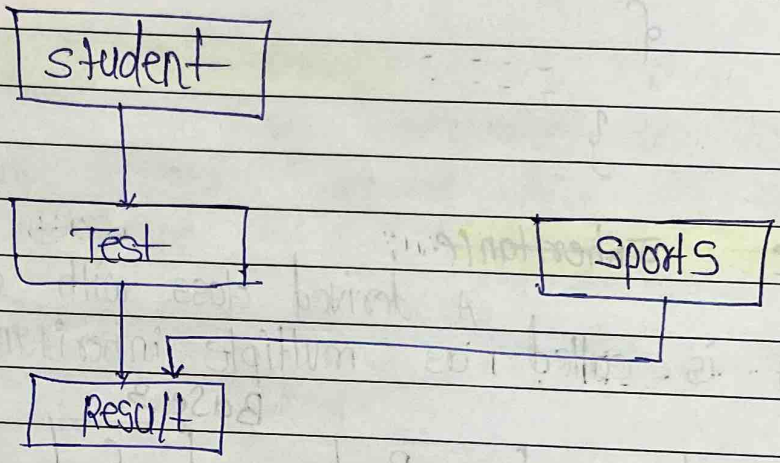
```
class A {---};
class B {-------};
class C {-------};
class D {-------};

class E : public A, public B, public
          C, public D ;
```

@curious_programmer

(4) Hybrid Inheritance

When sometimes situation occurred to apply two or more types of inheritance to design a program that is nothing but the Hybrid inheritance.



@curious_programmer

chapter 8

Templates

A template can be considered as a kind of macro. when an object of a specific type is defined for actual use, the template definition for that class is substituted with the required data type. since a template is defined with a parameter that would be replaced by a specified data type at the time of actual use of the class or function, the templates are sometimes called parameterized or functions.

The general format of a class Template is

```
template<class T>
class classname
{
    // ----
    // ---
    // ----
};
```

@curious_programmer

Example :

```
template<class T>
class vector
{
    T* v;
    int size;
public:
    vector(int m)
    { v = new T[size=m]; }
    vector (T* a)
    { ==
T operator *(vector &v)
{ ==
```

Example :

```
vector <int> v1(10);
vector <float> v2(25);
```

Class Templates with multiple parameters

We can use more than one generic data type in a class template. They are declared as a comma separated list within the template specification as show below.

```
Template < class T1, class T2, --->
  class className.
  {
    ----
    ----
    ----
  };
```

@curious-programmer

Function Templates

Like a class templates we can also define function templates that could be used to create a family of functions with different argument types. The general format of a function template is ;

```
Template <class T>
returnType functionname (argument of Type
    T)
{
}
```

Example :

```
Template <class T>
void swap (T&x, T&y)
{
    T temp = x;
    x = y;
    y = temp;
}
```

function Templates with multiples parameters

like template classes we can use more than one generic data type in the template statement, using a comma - separated list as shown below

```
Template <class T1, class T2, ---->
returntype function name ( arguments of type
    T1, T2, ----)
{
    - - - - -
    - - - - -
    - - - - ,
}
```