C++ : basic guidelines for beginners

Sumanta Pal,
Postdoctoral Fellow, Dept. of Physics, Univ. of Alberta
Email: sumanta@ualberta.ca

Git download details

git clone https://github.com/palSumanta/sapcppexamples.git

If you do not have git installed, install it from your linux repository.
For debian it is : sudo apt-get install git

Then go to the sapcppexamples directory.

If you do not have gcc/g++, install that too.

Language, Operating Systems, and Compilers

- The 'processor' is the brain of a computer. In a nutshell, it consists of a million of logic gates and digital switches.
- Any processor comes with a set of instruction set (A set of binary numbers which will do switching and mathematical operations plus other stuff in the processor), which is provided by the manufacturer.
- Programming using the instruction set given for a processor is usually called 'Assembly Level Programming or Low Level Programming/Language'.
    - Mnemonic code -> Binary code -> check if the codes are right -> check if the program runs properly.
- High Level Language, which is now used by a code developer, uses natural and understandable expressions.
    - High level languages hide all the complexities involved in putting together the pieces in the assembly language.

Continued .....

Language, Operating Systems, and Compilers

- High Level Languages are classified into two types:
  - Compiled Languages :
    - There is an intermediate ‘converter’ called a compiler which converts the statements in a program into binary instructions that the processor understands, apart from checking syntax errors.
    - C, C++ etc.
  - Interpreted languages :
    - Each statement in the code and then the corresponding instruction is converted and executed one after the other.
    - Java, Python etc.

Continued .....

Language, **Operating Systems**, and Compilers

- The Operating System is the manager of your computer.
- Linux is a family of open source Unix-like operating systems based on the Linux kernel, an operating system kernel first released on September 17, 1991, by Linus Torvalds. It is written in C and Assembly language [*wikipedia link*].
- It manages and controls the use of various resources in a computer.
- Linux Kernel : The Linux kernel is the main component of a Linux operating system (OS) and is the core interface between a computer's hardware and its processes. It communicates between the two, managing resources as efficiently as possible.
- The kernel is so named because, like a seed inside a hard shell, it exists within the OS and controls all the major functions of the hardware, whatever is our device : a phone, laptop, server, or any other kind of computer.

Continued .....

Language, Operating Systems, and **Compilers**

- List of Linux distributions (*wikipedia link*)
  - rpm based : RedHat, CentOS, Fedora etc.
  - debian based : Ubuntu etc.
  - 3rd party distribution : LinuxMint etc. [written based on Ubuntu's LTS package, and is tailored to user-friendliness for desktop users.
- I work in debian linux.
- Compiler : converts the statements of a program into a binary instructions that the processor understands, apart from checking syntax errors and linking to other libraries.
- *GCC/g++*, the GNU Compiler Collection, is used extensively for C, C++ programmes.
- The command line interface in Linux is called 'Terminal', 'Command Prompt' in Windows.

Contd .....

A “Hello World” program

Terminal

```
File  Edit  View  Search  Terminal  Help
sp@sp-ThinkCentre-M710e:~$ pwd
/home/sp
sp@sp-ThinkCentre-M710e:~$
```

Type your code in an editor : here it is ‘vi’

```
File  Edit  View  Search  Terminal  Help
#include <iostream>
using namespace std;

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

Compile the code : <compiler name> <main file name .cpp> <executable file name>

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ helloworld.cpp
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ls
a.out  helloworld.cpp
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ helloworld.cpp -o hellowor
ld
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ls
a.out  helloworld  helloworld.cpp
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ helloworld.cpp -o hellowor
ld.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ls
a.out  helloworld  helloworld.cpp  helloworld.exe
```

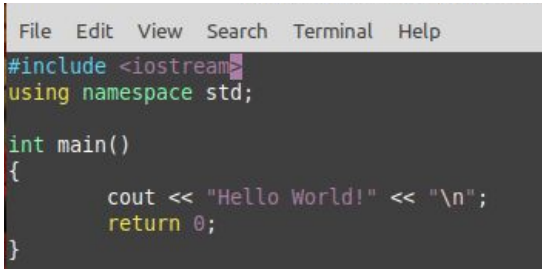
- $ gcc --version : shows gcc version
- $ gcc -v : shows its version with more details of its configuration.
- My version is 7.5.0 which supports c++17 (released in 2019).
  - More recent versions, 9.3, 10.1 are released in 2020. Version 11.0 is under development.
  - Choose latest version available through your linux OS package manager.
- Use an editor to write the code and Save it after that.
  - emacs, gedit/text editor, vi/vim, nano etc. (consumes less computer memory, command interface)
  - More professional editors nowadays : atom, eclipse, QtCreator etc. (consumes more computer memory, GUI interface)

Contd....

Execute the code : ./<executable file name>

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./helloworld.exe
Hello World!
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$
```

A “Hello World” program



#include <iostream> : input-output standard stream; This header file (preprocessor directive) contains definitions to objects like cin, cout, cerr etc. In any main program in C++ we need this header file.

namespace : provide a scope to identifiers inside it; we are not defining ‘cout, return, endl (alternate for “\n”) in the current scope, which is the main function.

‘using namespace std’ says computer to know the code for the ‘cout, cin’ functionalities and in which namespace they are defined.

If we do not define ‘namespace’ in the beginning, we can write this way: std::cout wherever it is needed . “::” is called ‘Scope Resolution Operator’.

“;” is required at the end of every line to end the line. Otherwise, compiler will give error.

‘Scope’ of a variable

Example 1

```
File  Edit  View  Search  Terminal  Help
#include <iostream>
using namespace std;

//global variable
int XX = 10;

// main program
int main()
{
        XX = 12;
        cout << "XX : " << XX << endl;
        cout << ::XX << ", " << XX << endl;

        return 0;
}
```

Output : XX : 12
12, 12

Example 2

```
File  Edit  View  Search  Terminal  Help
#include <iostream>
using namespace std;

//global variable
int XX = 10;

// main program
int main()
{
        int XX = 12;
        cout << "XX : " << XX << endl;
        cout << ::XX << ", " << XX << endl;

        return 0;
}
```

Output : XX : 12
10, 12

In general, the scope is defined as the extent up to which something can be worked with. In programming, the scope of a variable is defined as the extent of the program code within which the variable can we accessed or declared or worked with.

There are mainly two types of variable scopes: local variables and global variables.
Compiler gives precedence to local variable. If same variable is defined globally and locally (Example 2) then we can access the global variable using “::” (scope resolution operator).

Standard Indentation

File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

//main
int main()
{
        // variable decleration
        int XX;
        float YY;
        double ZZ;

        //assign a value
        XX = 1;
        YY = 2.0;
        ZZ = 2.2;

        // if statement
        if(XX > YY)
        {
                cout << "XX is greater than YY" << endl;
        }
        else
        {
                cout << "XX is less than YY" << endl;
        }

        // for loop
        for(int ii=0; ii<5; ii++)
        {
                cout << "ii : " << ii << endl;
        }

        return 0;
}
```

1. Indentation has to be maintained throughout the code.
2. It makes code clean and easily readable.
3. Necessary comments before each variable declaration, if statement etc. are very useful.
4. Variable name : I have given variable names such as XX, YY, ZZ. In any analysis code, a relevant name is always recommended.
5. Eg.: KineticEnergy or KE, Mass, Momentum etc. sometimes we also add ‘i’, or ‘f’, or ‘d’ etc. before variable name to remind us which variable is defined as integer, float or double. eg., int iCount, float fMass.

Data Types and Operators

- 'Variable' is a common term in programming language. For computer, it is just a memory slot.
- int iA = 999; when we define this a memory location is allocated for storing an integer at a specific address.
- The 'address' of a variable can be accessed through "&" operator.

File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

int main()
{
        int iA = 999;
        cout << "address of iA : " << &iA << endl;
        cout << "value/data of iA : " << iA << endl;

        return 0;
}
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example2.cpp -o example2.e
xe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example2.exe
address of iA : 0x7ffdba8e8164
value/data of iA : 999
```

- For every variable there is an address field and a data field.
- address is shown in hexadecimal number (0x represents hexadecimal number).
- Usage of 'address' will be more clear later when we'll just pass the address of a variable instead of passing its value.

Contd.....

Data Types and Operators

- A ‘bit’ is atomic : smallest unit of storage. Can store 0/1.
- A ‘bit’ is too small to use. A group of 8 bits make a ‘byte’. 1 byte = 8 bits. One byte can store one character, like ‘A’, ‘X’, ‘$’ etc.
- With 1 bit we have two patterns; 0 and 1.
- With 2 bits we have four patterns; 00, 01, 10, 11.
- With n bits we have 2^n patterns.
- Data type : char
  - 1 byte. It can make 2^8=256 different patterns. It can hold a number between 0 and 255 (0,1,2,...,255).
  - Really good for storing letters/characters.
  - Every character is defined by a number : like the alphabet A is 0x41 in Hex (65 in decimal)[read more here].

Contd.....

Data Types and Operators

- short : 2 bytes (short integer)
- int : 4 bytes (integer) [4 bytes = 32 bits; i.e. 2^{32} (= 4294967296) patterns. Range is -2147483648 to 2147483648]
- unsigned int : 4 bytes [range is 0 to 4294967295]
- long : 4 bytes (long integer)
- unsigned long :
- bool : 1 byte (boolean value true/false)
- float : 4 bytes (floating point)
- unsigned float :
- double : 8 bytes (double precision)
- long double : 8 bytes (long double precision)
- vector, pair, pointer, string, time variable etc. will be discussed later.

Contd.....

Data Types and Operators

- There are some special characters that are used in C/C++ which are usually not printed but have a special meaning. They are called 'Escape Characters or Escape Sequence'. [details]
- \t means horizontal tab, \v means vertical tab, \n means newline, \" double quote etc.

```
File  Edit  View  Search  Terminal  Help
#include <iostream>
using namespace std;

int main()
{
        cout << "How \"are \'you \?" << endl;
        cout << "How\tare\tyou\?\n";
        cout << "How\vare\vyou\?\n";

        return 0;
}
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example3.cpp -o example3.e
xe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example3.exe
How "are 'you ?
How     are     you?
How
   are
      you?
```

Contd.....

Data Types and Operators

- Mathematical operators :
  - Addition + [a=10; a = a+2; //12]
  - Subtraction - [a=10; a = a-2; //8]
  - Multiplication * [a=10; a = a*2; //20]
  - Division / [a=10; a = a/2; //5]
  - Modulo % [a=10; a = a%2; //0 gives the remainder]
- Compound Assignment operator :
  - += , -=, *=, \=, %=, >>=, <<=, &=, ^=, |=
  - a += 5; means a = a + 5;
- Increment and Decrement operator :
  - '++' is a increment operator [a=1; a++; \\ a becomes 2]
  - '--' is a decrement operator [a=1; a--; \\ a becomes 0]
  - int a; int b = 99; a = ++b; Now, a = 100 and b = 100
  - int a; int b = 99; a = b++; Now, a = 99 and b = 100

Data Types and Operators

- Relational operators :
  - == equal to (a==b) return false
  - != not equal to (a!=b) return true
  - > greater than (a>b) return true
  - < less than (a<b) return false
  - >= greater than or equal to (a>=b) return false
  - <= less than or equal to (a<=b) return false
- Logical operators :
  - ! logical inversion !(a>b) return false
  - && logical AND (a>b)&&(b<100) return true
  - || logicalOR (a<b)||(b<100) return false
  - a += 5; means a = a + 5;
- Conditional ternary operator :
  - Condition ? result1 : result2
  - a = 100, b=10. c = (a>b) ? a : b. So, c = a = 100
  - a = 100, b=10. c = (a<b) ? a : b. So, c = b = 10

Data Types and Operators

- bitwise operators :
  - These operators modify the bit pattern of a variable.
  - & AND Bitwise AND
  - | OR Bitwise inclusive OR
  - ^ XOR Bitwise exclusive OR
  - ~ NOT Unary complement (bit inversion)
  - >> SHR Shift bits right
  - << SHL Shift bits left

int a= 12, b=25; cout << (a&b); // output is 8 (in decimal).----

a = 12 = 0 0 0 0 1 1 0 0

b = 25 = 0 0 0 1 1 0 0 1

a&b = 0 0 0 0 1 0 0 0 = 8 (decimal)

a|b = 0 0 0 1 1 1 0 1 = 29 (decimal)

Contd.....

Data Types and Operators

Right Shift Operator

int a = 10; (in binary 1010)
int b = a >>1;
b = 5; (in binary 0101)

	2^5	2^4	2^3	2^2	2^1	2^0
a	0	0	1	0	1	0
b	0	0	0	1	0	1

Left Shift Operator

int a = 10; (in binary 1010)
int b = a <<1;
b = 20; (in binary 10100)

	2^5	2^4	2^3	2^2	2^1	2^0
a	0	0	1	0	1	0
b	0	1	0	1	0	0

This is useful when we handle binary data file which is written in a specific format in each experimental data. Unless we understand that specific format, real meaningful data can not be extracted out of that data binary file.

Contd.....

Data Types and Operators

- Typecasting operator :
  - float pi = 3.145
  - int ii = (int)pi; // ii = 3
  - Typecasting will be more obvious when we deal with pointers.

Control structure: if and else, loops, break, continue ...

```
if(condition)
{
        //condition is satisfied
        //do something
}
else
{
        //condition is not satisfied
        //do something
        //though it is absolutely
        //NOT necessary all the time
}
```

Contd .....

```
if(condition1)
{
        //condition is satisfied
        //do something
}
elseif(condition2)
{
        //condition2 is satisfied
        //do something
}
else
{
        // none of the conditions are satisfied
        //do something
        //if we skip it compiler will not complain
        //but it is better to have it for a clean code.
        //at least print a message
}
```

Control structure: if and else, loops, break, continue ...

for loop:

```
for (int ia=0; ia<101; ia++)
{
        cout << ia << “,” ;
}
```

There is an initialisation : int ia=0,
conditional part : ia<101, and
a statement which is executed : ia++

while loop:

While loop is repeated until the condition checked by the while statement is satisfied.

```
int a=0;
while (a<100)
{
        cout << a << “,” ;
        a++;
}
-----------------------
do
{
        cout << a << “,” ;
        a++ ;
}
while (a<100)
```

Control structure: if and else, loops, break, continue ...

Breaking in a for loop:

```
for (int ia=0; ia<101; ia++)
{
        cout << ia << “,” ;
        if ( ia == 50)
          {
              cout << “ia is 50” << ;
              break;
          }
}
```

‘break’ command will stop the loop.

continue keyword in a for loop:

```
for (int ia=0; ia<101; ia++)
{
        if ( ia == 50) continue; // skip the following line
        cout << ia << “,” ;
}
```

‘continue’ command will NOT stop the loop, but the following lines of code after it in the block are not executed.

Switch case structure

File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

int main()
{
        int a = 1;
        switch(a)
        {
                case 1:
                        cout << "a is 1 \n";
                        break;
                case 2:
                        cout << "a is 2 \n";
                        break;
                default:
                        cout << "value of a unknown \n";
        }

        return 0;
}
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example6.exe
a is 1
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$
```

Here, the value of 'a' is hard coded inside the code.
It is better to have an option to input it during execution of the code.

File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

int main()
{
        int a;
        cout << "Provide an integer input: ";
        cin >> a;

        switch(a)
        {
                case 1:
                        cout << "a is 1 \n";
                        break;
                case 2:
                        cout << "a is 2 \n";
                        break;
                default:
                        cout << "value of a unknown \n";
        }

        return 0;
}
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example6.exe
Provide an integer input: 1
a is 1
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example6.exe
Provide an integer input: 2
a is 2
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example6.exe
Provide an integer input: 3
value of a unknown
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example6.exe
Provide an integer input: 1.0
a is 1
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example6.exe
Provide an integer input: 1.5
a is 1
```

Function

```
File  Edit  View  Search  Terminal  Help
 1 #include <iostream>
 2 using namespace std;
 3
 4 int main()
 5 {
 6         float a,b;
 7         a = 10.;
 8         b = 5.5;
 9
10         cout << Add(a,b) << endl;
11
12         return 0;
13 }
14
15 float Add(float x, float y)
16 {
17         return (x+y);
18 }
```

The compiler converts the program by reading it line by line;
The compiler does not know 'Add' means at line 10 and thus return a compilation error.

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example7.cpp -o example7.e
xe
example7.cpp: In function 'int main()':
example7.cpp:10:10: error: 'Add' was not declared in this scope
  cout << Add(a,b) << endl;
          ^~~
```

Function

File Edit View Search Terminal Help

```
1  #include <iostream>
2  using namespace std;
3
4  //function prototype or forward decleration
5  float Add(float x, float y);
6
7  int main()
8  {
9          float a,b;
10         a = 10.;
11         b = 5.5;
12
13         cout << Add(a,b) << endl;
14
15         return 0;
16 }
17
18 //function actual definition
19 float Add(float x, float y)
20 {
21         return (x+y);
22 }
```

This time compiler knows what 'Add' function is when it reaches at line 13.

Here, two values are passed to the function and function returns one value.

What happens if we want our function to return more than one value?

Function

File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

//function prototype or forward decleration
float Add(float x, float y, float& z);
void AddSub(float x, float y, float&sum, float& diff);

int main()
{
        float a,b;
        a = 10.;
        b = 5.5;
        float r,c;
        r = Add(a,b,c);
        cout << "Add func. " << r << ", " << c << endl;

        float vSum, vDiff;
        AddSub(a,b, vSum, vDiff);
        cout << "AddSub func. " << vSum << ", " << vDiff << endl;

        return 0;
}

//function actual definition
float Add(float x, float y, float& z)
{
        z = x-y;
        return (x+y);

}

void AddSub(float x, float y, float&sum, float& diff)
{
        sum = x+y;
        diff = x-y;
}
```

In 'Add' function, function returns one value through 'return' command as its type is defined as float. Value of 'z' is passed through its address.

In 'AddSub' function, its type is void; so there is no 'return' command inside function body. Required values are passed through their addresses.

Remember, the address of any value is assigned by the OS and we can't change it. But we can access the address to see what value is assigned at that address or we can change the value for that variable at that address.

Function overloading

File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

// function overloading
int add(int x, int y)
{
        return (x+y);
}

float add(float x, float y)
{
        return (x+y);
}

int main()
{
        int a,b;
        a = 101; b = 100;
        float c,d;
        c = 101.3; d = 102.5;

        cout << add(a,b) << endl;
        cout << add(c,d) << endl;

        return 0;
}
```

An example of function overloading. Same function name 'add' accepts integer and float variables and do the addition.

Users no need to worry which function to use to add two integers and which one for float variables.

Arrays

Arrays are used to store a collection of values.
Say, I want to store the age of 5 people.
>> assigning a variable for each person, int age1, age2, ag3, .., age5.
>> using an array : int age[5];
This is an 1-dimensional array of size 5.
Assign values to the array : age[0] = 50, age[1]=20, age[2]=40, age[3]=33, age[4]=7.
Or, this way : int age[5] = {50,20,40,33,7};

2-dimensional array : int XX[5][6]. It is like a (5 x 6) matrix : 5 rows, 6 columns.

An array can have any number of dimensions but it has practical limitations imposed by the size of the physical memory in the computer.

Structures

Arrays are useful for storing a collection of values which are of similar type. Like, an integer array can only store integer values.

Structures are a collection of a different data types.
As an example, a structure named ‘student’ can store the age, height, name etc.

```
struct student {
  int age;
  float height;
  char  name[10];
};
```

Structures

File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

struct student {
        int age;
        float height;
        char stuname[10];
};

int main()
{
        student a,b;
        // a and b are variables of type student or
        // we call them objects of type student.
        // assign values to these objects

        cout << "enter a name ";
        cin.getline(a.stuname,10);
        cout << "enter height ";
        cin >> a.height;
        cout << "enter age ";
        cin >> a.age;

        //print assigned values
        cout << "Age: " << a.age;
        cout << ", name: " << a.stuname;
        cout<< ", height: " << a.height << endl;

        return 0;
}
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example9.cpp -o example9.e
xe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example9.exe
enter a name ABCD
enter height 131.2
enter age 22
Age: 22, name: ABCD, height: 131.2
```

Pointers

A pointer is a special variable which stores the address of a memory location.
A pointer to an integer is declared like: int *p;
int a; //just doing this a memory location will be created for ‘a’ with some random value.

address	data	variable
5000	56738	a

a=100; // a value is assigned

address	data	variable
5000	100	a

int *pa; // just defined a pointer to an integer. A memory location will be created

address	data	variable
2600	23458	pa

int *pa = &a; //now a specific address, here address of variable ‘a’ is passed to the pointer

address	data	variable
2600	5000	pa

Pointers

Only addresses can be passed to a pointer.

We can not initialise a random value to a pointer. If we do ‘int *p = 100;’ then compiler will complain.

The value of ‘a’ was assigned to 100 earlier. We want to change it to 200 but using the pointer which has the address of ‘a’.
*pa = 200; [previously did : int *pa = &a;]

& is called the reference operator and * is called the dereference operator.

We can also define pointer to a pointer. int *ppa = &pa.

Contd.....

Pointers

If there is a data type like int, float etc., before a '*', then the star denotes that the variable next to it is a pointer.
If not, then the star should be read as "the value at memory location pa is …"

```
int var = 10;   //variable declaration
int *pvar;      // pointer defined
pvar = &var; //assign address of 'var' to pointer pvar
int *pvar = &var; //same thing but done in one line

*pvar = &var; // WRONG here *pvar reads the value at memory location
*pvar = 20; // CORRECT here we change the value at memory location from 10 to 20.
```

String

```cpp
#include <iostream>
#include <string>

using namespace std;

int main () {

   string str1 = "Hello";
   string str2 = "World";
   string str3;
   int  len ;

   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
   cout << "str1 + str2 : " << str3 << endl;

   // total length of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

String is the modified version of character array in C. [char str1[10]={'H','e','l','l','o'};]

str1.begin()
str1.end()
str1.erase(size_t pos, size_t len)

```
str3 : Hello
str1 + str2 : HelloWorld
str3.size() :  10
```

Time variable

- The C++ standard library does not provide a proper date type for time.
- C++ inherits the structs and functions for date and time manipulation from C.
  - To access date and time related functions and structures, you would need to include <ctime> header file in your C++ program.
- There are four time-related types: clock_t, time_t, size_t, and tm.
  - The types - clock_t, size_t and time_t are capable of representing the system time and date as some sort of integer.
  - The structure type tm holds the date and time in the form of a C structure having the following elements
    -

Time variable

- The structure type tm holds the date and time in the form of a C structure having the following elements

```
struct tm {
   int tm_sec;   // seconds of minutes from 0 to 61
   int tm_min;   // minutes of hour from 0 to 59
   int tm_hour;  // hours of day from 0 to 24
   int tm_mday;  // day of month from 1 to 31
   int tm_mon;   // month of year from 0 to 11
   int tm_year;  // year since 1900
   int tm_wday;  // days since sunday
   int tm_yday;  // days since January 1st
   int tm_isdst; // hours of daylight savings time
}
```

Time variable

- time_t time(time_t *time); This returns the current calendar time of the system in number of seconds elapsed since January 1, 1970. If the system has no time, .1 is returned.
- char *ctime(const time_t *time); This returns a pointer to a string of the form *day month year hours:minutes:seconds year\n\0*.
- struct tm *localtime(const time_t *time); This returns a pointer to the tm structure representing local time.
- clock_t clock(void); This returns a value that approximates the amount of time the calling program has been running. A value of .1 is returned if the time is not available.
- char * asctime (const struct tm * time); This returns a pointer to a string that contains the information stored in the structure pointed to by time converted into the form: day month date hours:minutes:seconds year\n\0
- struct tm *gmtime(const time_t *time); This returns a pointer to the time in the form of a tm structure. The time is represented in Coordinated Universal Time (UTC), which is essentially Greenwich Mean Time (GMT).
- time_t mktime(struct tm *time); This returns the calendar-time equivalent of the time found in the structure pointed to by time.
- double difftime (time_t time2, time_t time1); This function calculates the difference in seconds between time1 and time2.
- size_t strftime(); This function can be used to format date and time in a specific format.

Time variable

example13.cpp

```cpp
#include <iostream>
#include <ctime>

using namespace std;

int main() {
   // current date/time based on current system
   // time_t returns the current calendar time of the system
   // in number of seconds elapsed since January 1, 1970.
   //If the system has no time, .1 is returned.
   time_t now = time(0);
   cout << "Number of sec since January 1,1970:" << now << endl;

   // convert now to string form
   char* dt = ctime(&now);

   cout << "The local date and time is: " << dt << endl; //day-month-date hh:mm:sec year

   // convert now to tm struct for UTC
   tm *gmtm = gmtime(&now);
   dt = asctime(gmtm);
   cout << "The UTC date and time is:"<< dt << endl; //day-month-date hh:mm:sec year

   //format time using tm structure
   // current date/time based on current system

   tm *ltm = localtime(&now);

   // print various components of tm structure.
   cout << "Year:" << 1900 + ltm->tm_year << endl;  //year since 1900
   cout << "Month: "<< 1 + ltm->tm_mon<< endl;
   cout << "Day: "<<  ltm->tm_mday << endl;
   cout << "Time: "<< 1 + ltm->tm_hour << ":";
   cout << 1 + ltm->tm_min << ":";
   cout << 1 + ltm->tm_sec << endl;

   return 0;
}
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example13.cpp -o example13
.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example13.exe
Number of sec since January 1,1970:1588737280
The local date and time is: Tue May  5 21:54:40 2020

The UTC date and time is:Wed May  6 03:54:40 2020

Year:2020
Month: 5
Day: 5
Time: 22:55:41
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$
```

File I/O

- A new header is required to perform file input/output : <fstream>
  - <ifstream> this allows only to read file
  - <ofstream> this allows only to write file
  - <fstream> this allows both : read & write
- File Object : fstream outFile.
  - fstream is a class in C++. treat it like other variable type int/float etc. what we have done before.
  - Consider 'outFile' is an object of type fstream.
- Opening a file : outFile.open("outf.txt", ios::out | ios:app);
  - 1st argument is a file name
  - 2nd argument is a combination of some flags.
- Flags :
  - ios::out open file for writing
  - ios::in open file for reading
  - ios::app append file
  - ios::trunc delete file (if it already exists) and open
  - A combination of them is possible unless and until they conflict each other. Say, ios:app and ios::trunc is not possible together.

Contd.....

File I/O

File Edit View Search Terminal Help

```
1  #include <iostream>  //input-output standard stream lib for c++
2  #include <iomanip>   //input-output manipulator lib
3  #include <fstream>   //file input-output lib
4  #include <string>
5  using namespace std; //standard library namespace
6
7  int main()
8  {
9          //first write a file
10         fstream outFile;
11         //fstream does both, read and write.
12         //if it confuses you to distinguish which file is written
13         //and which is read we can do this also
14         //ofstream outFile;
15         outFile.open("outF.txt",ios::out|ios::trunc);
16         // outFile is an object of type fstream
17         // 'open' func. allows to open the file
18         // ios::trunc will delete a file with same name if exists
19         // and open the file
20         // ios::out will allow to write into this file
21         outFile << "Day" << setw(10) << "Month" << setw(10) << "Year" << endl;
22         outFile << "Sun" << setw(10) << "May" << setw(10) << 2020 << endl;
23         outFile.close(); // close the file once write is done.
24
25         //reading a file is only possible if you know what is there
26         //inside the file and the format
27         //Let's open the file
28         fstream inFile; // or ifstream inFile
29         inFile.open("outF.txt", ios::in);
30         //pretend that we don't know what is written in this file
31         //so read every line as a whole until we reach end of file
32         string input;
33         while(!inFile.eof())
34         {
35                 getline(inFile,input); //getline will read each line
36                 if(inFile.eof()) break;//check commenting out this
37                 cout << input << " -- \t" << "size of input string: " << input.size() <<endl;
38
39         }
40         inFile.close();
41         return 0;
42 }
43
                                                          1,39          All
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example10.cpp -o example10
.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example10.exe
Day      Month      Year --      size of input string: 23
Sun        May      2020 --      size of input string: 23
```

Most of the time we read ASCII file like this.

Binary file is a special file, and we can't read it meaningfully unless we know its format. "ios::binary" is required while opening it.

In high energy physics, we use ROOT software, and a root file has a specific structure.

Classes

- Class definition in C++ is its main ingredient.
- Data abstraction and object oriented programming are the main features of C++.
- It may strike for a beginner that the use of C++ is a bit of work around and that most of things could be achieved with C itself. The fact is true without any doubt.
- But the main goal here is not just the achievement but the way it is achieved and the time it takes to achieve. Until we write a program that deserves the power of C++ it is hard to convince oneself.
- The object oriented methodology allows to express here abstract idea very simply. It does not mean that it improves program performance, and that depends on many things.

Classes :

- In ‘struct’ all members are public. But in ‘class’ there are three types: private, public, and protected. By default, any member is private.

```
class vector
{
        private:
        int x,y,z; // member variables
        public:
        void Initialise(int a, int b, int c); // member function
};
void vector::Initialise(int a, int b, int c) {x=a; y=b; z=c;} // remember “::”
int main()
{
    vector v;
    v.Initialise(1,2,3);
     cout  << v.x  << endl;  // Is it correct? Answer is NO. Because, ‘x’ is defined in the scope of class vector; it is a private member variable of class vector. The object ‘v’ to class vector inside main function can not access it.
}
```

Classes : private, protected, public keyword

keywords	accessible by
private :	members of the same class or friends
protected:	members of the same class, friends, and derived classes
public:	all

```
class aclassname
{
        private:
                //define members which are private
        public:
                //define members which are public
        protected:
                //define members which are protected
};
```

Classes : introduce some methods inside a class

```
class Vector
{
        int x,y,z; // default these are private members to this class
        public:
                void Initialise(int a, int b, int c);
                float Length(); // this is a method. This will calculate the length of the vector
};

void Vector::Initialise(int a, int b, int c) { x=a; y=b; z=c;}

float Vector::Length() { return sqrt(x*x + y*y +z*z); }
```

Classes : constructor & destructor

- In the previous example we had an explicit method called ‘Initialise’ to set the values of the vectors.
- Without calling ‘Initialise’ first in ‘main()’ function, if we call the other method ‘Length’, it will return junk value.
- Therefore, it is good that all classes have such a method which initialises the variables properly and it is better if such crucial methods are not avoided accidentally. This feature is built in C++ using ‘constructors’.
- A ‘constructor’ can be called as an initialisation method of a class.
  - The ‘constructor’ method has the same name of the class.
  - It takes in variables as parameters just like any other function.
  - A ‘constructor’ can be called only at the time of construction of an object but not anywhere else.
- A ‘destructor’ is the opposite of the ‘constructor’. It is called when an object is deleted.
  - The ‘destructor’ is called before the end of the program automatically even without an explicit call to it.
  - Destructor overloading is not allowed in C++.

Classes : constructor & destructor

```
File  Edit  View  Search  Terminal  Help
#include <iostream>
#include <cmath>
using namespace std;

class Vector
{
        int x,y,z; //default it is private member of the class
        public:
           Vector(int a, int b, int c); //Constructor1 earlier we used Initialise here
           Vector();//Constructor2
           ~Vector();//Destructor
           //methods
           float Length();
};
Vector::Vector(int a, int b, int c)
{
        x=a; y=b; z=c;
}
Vector::Vector()
{
        x=0; y=0; z=0;
}
Vector::~Vector()
{
        cout << "Destructor for Vector " << endl;
}
float Vector::Length()
{
        return ( sqrt(x*x + y*y +z*z) );
}

int main()
{
        Vector v1(1,2,3); //v1 is an object to class Vector using constructor1
        Vector v2; //v2 is another object to class Vector using constructor2.

        cout << "Length of v1: " << v1.Length() << endl;
        cout << "Length of v2: " << v2.Length() << endl;

        return 0;
}
~
~
"example11.cpp" 41L, 803C written                              2,11          All
```

A class can have many constructors each taking different arguments, similar to function overloading. Here, we have two constructors, same name but different arguments.

It is important to notice that if any constructor has no parameters then do not use the brackets when an object is created from it.

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example11.cpp -o example11
.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example11.exe
Length of v1: 3.74166
Length of v2: 0
Destructor for Vector
Destructor for Vector
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$
```

Classes : object and pointer to a class

```
File  Edit  View  Search  Terminal  Help
 1 #include <iostream>
 2 #include <cmath>
 3 using namespace std;
 4
 5 class Vector
 6 {
 7         int x,y,z; //default it is private member of the class
 8         public:
 9            Vector(int a, int b, int c); //Constructor1 earlier we used Initialise here
10            Vector();//Constructor2
11            ~Vector();//Destructor
12            //methods
13            float Length();
14 };
15 Vector::Vector(int a, int b, int c)
16 {
17         x=a; y=b; z=c;
18 }
19 Vector::Vector()
20 {
21         x=0; y=0; z=0;
22 }
23 Vector::~Vector()
24 {
25         cout << "Destructor for Vector " << endl;
26 }
27 float Vector::Length()
28 {
29         return ( sqrt(x*x + y*y +z*z) );
30 }
31
```

```
29         return ( sqrt(x*x + y*y +z*z) );
30 }
31
32 int main()
33 {
34         Vector v1(1,2,3); //v1 is an object to class Vector using constructor1
35         Vector v2; //v2 is another object to class Vector using constructor2.
36
37         cout << "Length of v1: " << v1.Length() << endl;
38         cout << "Length of v2: " << v2.Length() << endl;
39
40         // now do the same using pointer instead of object to a class
41         Vector *v3 = new Vector(1,2,3);
42         // 'new' keyword returns a pointer to the class.
43         // this is the way we have to define a pointer to a class
44         Vector *v4 = new Vector();
45         // see now we need '()' at the end in compare to object definition
46
47         cout << "Length of v3: " << v3->Length() << endl; // Look '->' sign
48         cout << "Length of v4: " << v4->Length() << endl; // instead of '.' sign
49
50         delete v3; //without this destructor will not be called explicitly
51         delete v4; //this may result 'memory leaks' in large program
52
53         return 0;
54 }
                                                                    54,1          Bot
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example11.cpp -o example11
.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example11.exe
Length of v1: 3.74166
Length of v2: 0
Length of v3: 3.74166
Length of v4: 0
Destructor for Vector
Destructor for Vector
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example11.cpp -o example11
.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example11.exe
Length of v1: 3.74166
Length of v2: 0
Length of v3: 3.74166
Length of v4: 0
Destructor for Vector
Destructor for Vector
Destructor for Vector
Destructor for Vector
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$
```

When compiled without 'delete v3 and delete v4'.

Three new keywords:
new : to create a pointer
-> : to access members
delete : to delete and freed memory used by the pointer

Classes: Inheritance

- One of the most important concepts in object-oriented programming.
- Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.
- Opportunity to reuse the code functionality and fast implementation time.
- When a new class inherits the members of an existing class, then new class is called a 'derived class' and the from which it is inherited is called the 'base class'.
- Think this way : the idea of inheritance implements the 'Is A' relationship.
    - Mammal 'Is A' animal. Dog 'Is A' mammal; hence dog 'Is A' animal too.
    - Geant4 is a well extensive simulation software used in high energy physics, nuclear physics, space science, medical physics. Many physics processes are inherited from a base class. Say, Electromagnetic Physics is a base class; then Photoelectric Effect, Compton Scattering, Pair production will all inherit from Electromagnetic Physics class.
    - ROOT is a huge data analysis software used in High Energy Physics. Think about main histogram class which is a base class; then 1D histogram, 2D histogram are all inherited class from the base class.

Classes: Inheritance

```
File  Edit  View  Search  Terminal  Help
#include <iostream>
using namespace std;

// base class
class shape
{
        protected:
                float length, breadth;
        public:
                shape()//constructor
                {
                        cout << "In shape constructor 0" << endl;
                        length = 0.; breadth = 0.;
                }

                shape(float l, float b)//constructor
                {
                        cout << "In shape constructor 1" << endl;
                        length = l; breadth = b;
                }
                ~shape();//destructor
};
shape::~shape()
{
        cout << "shape destructor" << endl;
}
//derived class
class square:public shape
{
        public:
                square(float a)
                {
                        cout << "square constructor" << endl;
                        length = a; breadth =a;
                }
                ~square();//destructor
                float area() //a method
                {
                        return length*breadth;
set number
```

```
File  Edit  View  Search  Terminal  Help
                        length = a; breadth =a;
                }
                ~square();//destructor
                float area() //a method
                {
                        return length*breadth;
                }
};
square::~square()
{
        cout << "square destructor" << endl;
}
class rect:public shape
{
        public:
                rect(float a, float b):shape(a,b) //constructor
                {
                        cout << "rect constructor" << endl;
                }
                ~rect();//destructor
                float area()//a method
                {
                        return length*breadth;
                }
};
rect::~rect()
{
        cout << "rect destructor" << endl;
}

int main()
{
        square sq(2);
        rect r(3,4);
        cout << "square area = " << sq.area() << endl;
        cout << "rect area = " << r.area() << endl;

        return 0;
}
```

Syntax:

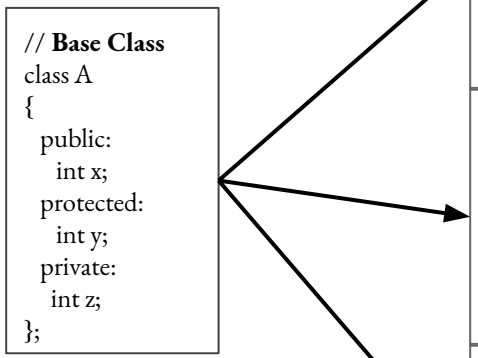
class subclassName : access_mode baseclassName {
  // body of subclass
};

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example12.cpp -o example12
.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example12.exe
In shape constructor 0
square constructor
In shape constructor 1
rect constructor
square area = 4
rect area = 12
rect destructor
shape destructor
square destructor
shape destructor
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$
```

Multiple and Multilevel inheritance are also possible.

Classes : Inheritance : modes of inheritance

There are three modes : Public, Protected, Private



```
// Base Class
class A
{
  public:
    int x;
  protected:
    int y;
  private:
    int z;
};
```

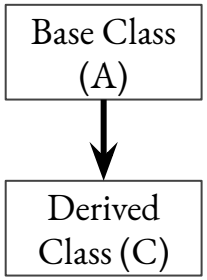
// derived Class : mode public
class B : public A
{
// x public
// y protected
// z can't be accessed from B
};
	public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.

// derived Class : mode protected
class C : protected A
{
// x protected
// y protected
// z can't be accessed from C
};
	public member of the base class will become protected in the derived class and protected members of the base class will remain as protected in derived class.

// derived Class : mode private
class D : private A
{
// x private
// y private
// z can't be accessed from D
};
	public & protected member of the base class will become private in the derived class.

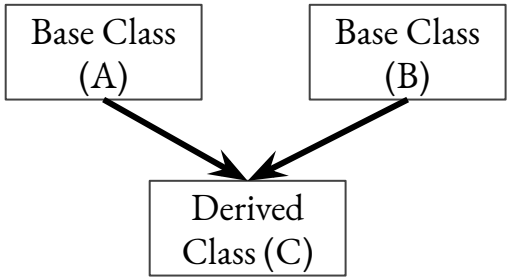
Classes : Inheritance : levels of inheritance

Single Inheritance



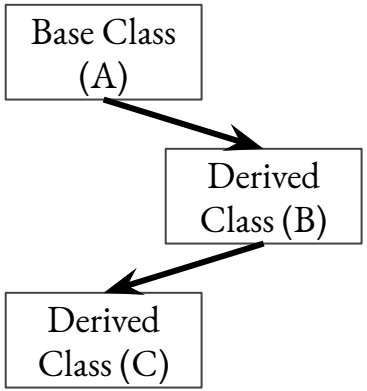

```
class C : public A
{
  //class body
};
```

Multiple Inheritance




```
class B : public A, public B
{
  //class body
};
```

Multilevel Inheritance




```
class B : public A
{
  //class body
};
class C : public B
{
  //class body
};
```

There are Hierarchical Inheritance and Hybrid Inheritance.

Vector : a dynamic array

example14.cpp

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
  vector<int> myvector;
  //vector<int> myvector[]; //we can also have vector array

  for (int i=1; i<=5; i++) myvector.push_back(i);

  cout << "size of myvector : " << myvector.size() << endl;

  cout << "myvector contains:" << endl;
  for (vector<int>::iterator it = myvector.begin(); it != myvector.end(); ++it)
    {
      cout << ' ' << *it << '\n';
    }

  cout << "same as above but different way " << endl;
  for (int it = 0; it< (myvector.size()); ++it)
    {
      cout << ' ' << myvector.at(it) << '\n';
    }

  return 0;
}
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example14.cpp -o example14
.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example14.exe
size of myvector : 5
myvector contains:
 1
 2
 3
 4
 5
same as above but different way
 1
 2
 3
 4
 5
```

Friend function inside a class accessing class private member

The private and protected member can be accessed by friend functions and friend classes.

File Edit View Search Terminal Help

```
        cout << "square destructor" << endl;
}
class rect:public shape
{
        public:
                rect(float a, float b):shape(a,b) //constructor
                {
                        cout << "rect constructor" << endl;
                }
                ~rect();//destructor
                float area()//a method
                {
                        return length*breadth;
                }
                //
                //friend function
                friend float GetLength(rect r);
};
rect::~rect()
{
        cout << "rect destructor" << endl;
}
//
float GetLength(rect r)
{
        return r.length;
}

int main()
{
        square sq(2);
        rect r(3,4);
        cout << "square area = " << sq.area() << endl;
        cout << "rect area = " << r.area() << endl;

        cout << "The length of rect = " << GetLength(r) << endl;

        return 0;
}
"example12.cpp" 82L, 1328C written
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ example12.cpp -o example12
.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./example12.exe
In shape constructor 0
square constructor
In shape constructor 1
rect constructor
square area = 4
rect area = 12
The length of rect = 3
rect destructor
shape destructor
rect destructor
shape destructor
square destructor
shape destructor
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$
```

Friend class

Like friend functions, there can be friend classes too. If a class is declared as a friend, the friend class can access the members of the class that declared as a friend (not vice-versa). In below example, the GetArea(square s) is able to access the member s.Length only because the rect class is defined as friend in square class.

```
File  Edit  View  Search  Terminal  Help
#include <iostream>
using namespace std;

class square
{
        float Length; // this is private by default
        public:
                square(float a) //constructor
                {
                        Length = a;
                }
                ~square(); //destructor

                // a method, just a print statement
                void printvalues(){
                        cout<<"l " << Length << endl;
                }

                // give access to rect class
                friend class rect;

                // is it allowed? Answer is NO
                //float getarea(rect);
};

square::~square(){}

class rect
{
        float length, breadth; //private
        public:
                rect(float a, float b) // constructor
                {
                        length = a;
                        breadth = b;
                }

                ~rect();
```

```
                //Methods
                float GetArea();
                float GetArea(square);
};

rect::~rect(){}

//define methods
float rect::GetArea()
{
        return length*breadth;
}

float rect::GetArea(square s)
{
        length = s.Length;
        breadth = s.Length;
        return GetArea();
}

int main()
{
        square s(2);
        rect r(3,4);

        cout << "Area rect : " << r.GetArea() << endl;

        cout << "Area square : " << r.GetArea(s) << endl;

        //cout << "Area square : " << s.GetArea() << endl;
        // square can not access GetArea() defined inside rect class
        // compiler will show this error:
        // friendclass.cpp:66:32: error: 'class square' has no member named 'GetArea'
        // cout << "Area square : " << s.GetArea() << endl;
        //
        s.printvalues();

        return 0;
}
                                                        72,1-8
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ friendclass.cpp -o friendc
lass.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./friendclass.exe
Area rect : 12
Area square : 4
l 2
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$
```

Try with pointer.
square s(2) will be replaced by,
square *s = new square(2).
s.printvalues () will be
s->printvalues()

Polymorphism and Virtual member

Polymorphism : The type compatibility of the pointer of the base class with the derived class is called polymorphism. That is the pointer of the base class can point to the pointer of the derived class but not vice-versa.

Virtual Member : the base class pointer can not still access the members of the derived class.
Like, s->area() is allowed, but sh2->area() is not allowed.
Now, we can move the 'area' method into base class; but if we introduce another class, say triangle, its area is calculated is differently.
This is solved by 'virtual member'. If a particular method is common to all the derived classes but have different implementations, a virtual method should be defined in the base class.

Look at next example slide to make it clear

Polymorphism and Virtual member

```
File  Edit  View  Search  Terminal  Help
 1 #include <iostream>
 2 #include <fstream>
 3 using namespace std;
 4 class shape{
 5 protected:
 6   float length, breadth;
 7 public:
 8   shape(){
 9     cout<<"In Shape ctor 0"<<endl;
10     length=0;
11     breadth =0;
12   }
13   shape(float l, float b){
14     cout<<"In Shape ctor 1"<<endl;
15     length=l;
16     breadth =b;
17   }
18   void PrintValues(){
19     cout<<"Length: "<<length<<" Breadth: "<<breadth<<endl;
20   }
21   virtual float area(){
22   return -1;
23   }
24 };
25 class square:public shape{
26 public:
27   square(float a){//ctor
28     cout<<"In square ctor"<<endl;
29     length = a;
30     breadth=a;
31   }
32   float area(){
33     return  length*breadth;
34   }
35 };
36 class tri:public shape{
37 public:
38   tri(float a, float b){//ctor
39     cout<<"In triangle ctor"<<endl;
"virtual.cpp" [dos] 59L, 1128C written
```

```
39     cout<<"In triangle ctor"<<endl;
40     length =   a;
41     breadth=  b;
42   }
43   float area(){
44     return  0.5*(length*breadth);
45   }
46 };
47 int main() {
48   square *s=new square(2);
49   tri *t=new tri(2,3);
50   shape *sh1 =s;  //Polymorphism
51   shape *sh2 =t;  // ,,
52
53   cout<<"square area: "<<sh1->area()<<endl;
54   cout<<"tri area: "<<sh2->area()<<endl;
55
56   cout << s->area() << ", " << t->area() << endl;
57
58   return 0;
59 }
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ virtual.cpp -o virtual.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./virtual.exe
In Shape ctor 0
In square ctor
In Shape ctor 0
In triangle ctor
square area: 4
tri area: 3
4, 3
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$
```

Compiling code :

- So far, we have discussed functionality of the C++ programming language.
- Now, we need to understand organising the source code, header file, executable binary file in a more structured way.
- What are *.o or *.so files?
- What is ‘Make’ command?
- What about ‘cmake’?

Compiling code : header file, objects and linking

Add.h
```
#ifndef ADD_H
#define ADD_H

int Add(int x, int y); //just define the function

#endif
```

Add.cpp
```
//for this simple function we don't
//need <iostream> or std namespace
//
#include "Add.h"

int Add(int x, int y)
{
   return (x+y);
}
```

mainAdd.cpp
```
#include <iostream>
#include "Add.h"
using namespace std;

int main()
{
  int a=10;
  int b=4;
  int c = Add(a,b);
  cout << "output of Add fn. " << c << endl;

  return 0;
}
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ mainAdd.cpp Add.cpp -o mai
nAdd.exe
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./mainAdd.exe
output of Add fn. 14
```

Function forward declaration is done by in the main program when we do #include "Add.h".

In a big program, we don't want to repeat writing such a function in the main program or in a different source file.

What we do we generate an object file for the function routine. That object file can be called any time by any program just my linking to the main program.

g++ -c -fPIC Add.cpp
It will create 'Add.o' file.

Then compile main program as g++ mainAdd.cpp Add.o -o mainAdd.exe

Compilation of a big program, like CERN-ROOT, takes about 2 hours in a laptop.

Compiling code : header file, objects and linking

Add.h | Add.cpp | mainAdd.cpp | Sub.h | Sub.cpp | mainAddSub.cpp

```
#include <iostream>
#include "Add.h"
#include "Sub.h"
using namespace std;

int main()
{
  int a=10;
  int b=4;
  int c = Add(a,b);
  cout << "output of Add fn. " << c << endl;
  int d = Sub(a,b);
  cout << "output of Sub fn. " << d << endl;

  return 0;
}
```

```
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/Desktop/my_cpp_lecture
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ g++ mainAddSub.cpp -o mainAddSub.exe -L. -lmyMath
sp@sp-ThinkCentre-M710e:~/Desktop/my_cpp_lecture$ ./mainAddSub.exe
output of Add fn. 14
output of Sub fn. 6
```

Make another file called “Sub.h” which returns (x-y).
Then create an object file for Sub.cpp -> g++ -c -fPIC Sub.cpp. It’ll create Sub.o.

Now, we’ll create a math library “myMath” which contains Addition & Subtraction function into it.
command: g++ -shared -o libmyMath.so Add.o Sub.o

Now, we have a library of functions, compiled and ready to be called by any programs. What we have to do is just to add necessary header file in main program and linking libraries during compilation.

Compilation command:
g++ mainAddSub.cpp -o mainAddSub.exe -L. -lmyMath
-- ‘-l’ before myMath tells the compiler to link to the library myMath. The prefix ‘lib’ and file extension .so have been dropped.
Flag ‘-L.’ says the library .so file exists in the current directory. If it is not, then we need -L/<dirpath of library *.so file>

Are you getting “error while loading shared libraries: libmyMath.so: cannot open shared object file: No such file or directory”?

Do this : export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<dirname>

Useful links

- GNU Make : https://www.gnu.org/software/make/manual/make.html
- GNU Make easy example : https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html#zz-2.1
- C++ variable details: http://www.cplusplus.com/doc/tutorial/
- Cmake main website : see tutorial in this page : https://cmake.org/documentation/