**C++ Notes:**

**Parts of a c++ program:**

1. **Header files**
2. **Namespace**
3. **Main Function**
4. **External Functions/Classes are optional**

**Header files:**

>>Header file are libraries that have classes and functions

>>The moment you include a library into your program, all of it’s code becomes part of your program code.

>>Always try to use <bits/stdc++.h> if possible.

>> <algorithm> package offers useful functions like reverse(), \_\_gcd()max(),min(),sort()

Reverse() and sort() work for all collections like string, array, STls

>> <ctype> is a substitute for <string.h>

>> <string> and <string.h> are different.

>><stdlib>=abs(),trunc(),ceil(),trunc()

>>You can create your own header files too.

All you need to do is write some functions in txt file and save it in the current directory with .h extension.

After that use it as a normal header file.

**Cin, Cout:**

>>cin can recognize separate inputs separated by space or newline.

>>after completion of execution of a cin statement cursor moves to newline.

>>contrary to this after completion of a cout statement cursor still remains in the same line

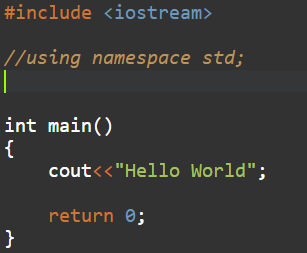
>>cin by default in case of string will only read one word, hence instead use getline(cin,string\_name) to read the whole line instead of normal cin for string.

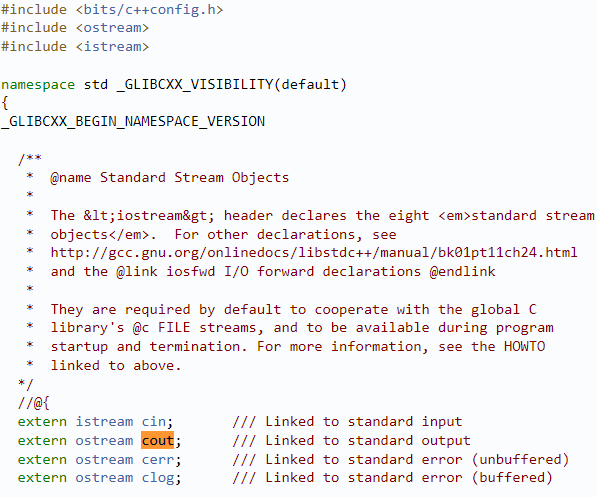
**Namespace:**

>>Namespace is a like logical partition of code space with it’s own scope. Used for grouping code

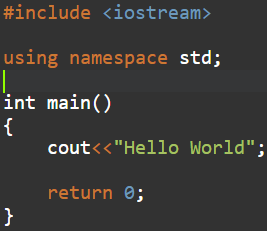
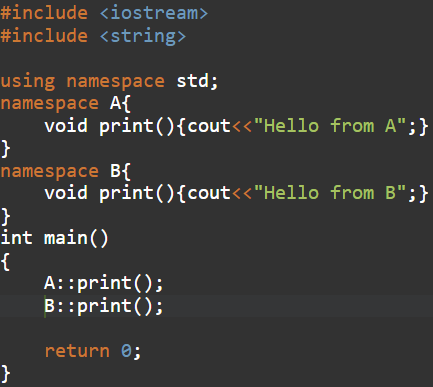
>>The code inside the namespace won’t work outside the namespace scope.

>>For example take the below code, now it’s known that cout operator is part of iostream library. But the compiler still says cout is not declared in this scope for the below code.



So why this error? Well it because if you take a look into the code of iostream.h

So as you can see the cout exisits in the std namespace in iostream. So unless you import the std namespace for your entir program using the following syntax: using namespace std; the cout will remain unrecognized

You can also use scope resolution operator :: instead of importing the whole scope

**Precision:**

>>int/int returns only int, only float/int will return exact value in float

Float/int=float, float/float=float, int/float=float.

>>printf("\n%0.6f",res2); //will print float precise upto 6 decimal points

>>cout<<setprecision(7)<<res2; //<iomanip.h> required

**Data Types:**

>>**Primitive-** int, char float, double

**Derived-** array, pointer

**User Defined-** union, Structure, Class, enum

>>**Modifiers-** long, unsigned(will only allow +ve values)

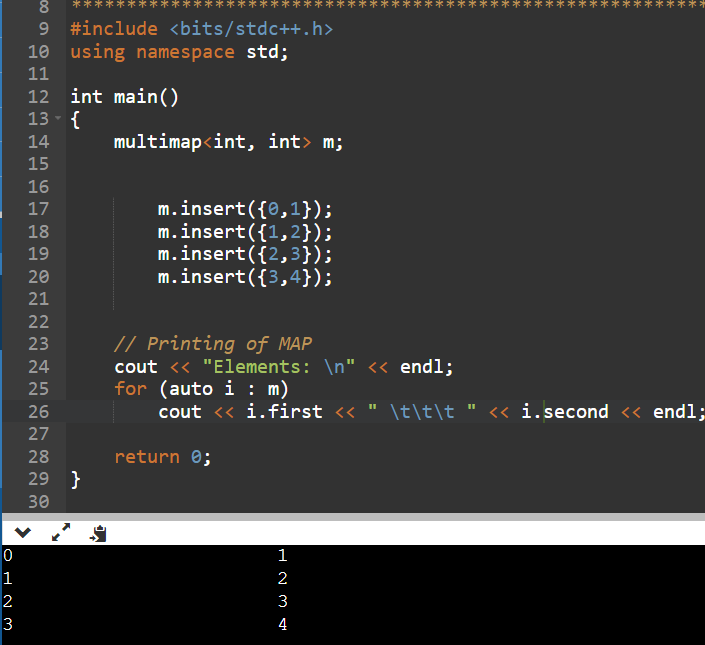
Note:- OOPs concept are applicaple start from class, inheritance is not allowed in Struct

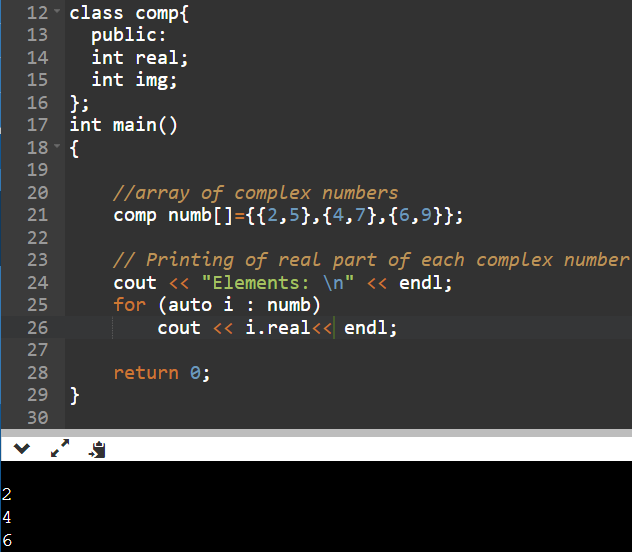
**Auto:**

Auto is a new feature of cpp 20, auto is an advance data type which allows to compiler to determine the data type of a variable depending on the value provided for it by the user.

Below are some uses of auto:

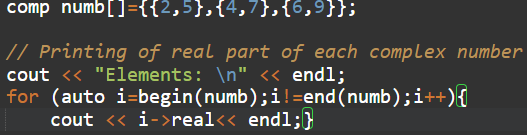
1. As datatype of variables
2. As return type of function and data type of it arguments
3. In for each loop as an object to traverse a map





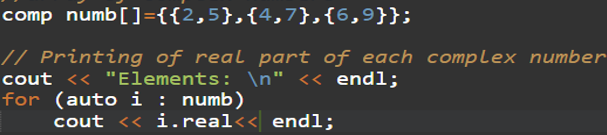
1. As an iterator to traverse map/queue

**Note:-** IF you provide stl or object array address to auto i, then I will behave as a pointer iterator



Or

If you directly assign object the auto I will behave as temp object

****

**Array:**

>>Arrays are static, means that size is fixed.

>>The size of the array should be know at the time of declaration, For e.g.

**Some valid Declarations are like:**

-int n;

cin>>n;

int ar[n];

-int n;

cin>>n;

int \*ar;

ar=(int \*)malloc(n\*sizeof(int));

-char ar[]="arun";

int br[]={1,2,3};

-int ar[2];

char br[2];

**Invalid declarations:**

-int ar[];

char br[];

Note: Array name, in above case ‘ar’ is equivalent to array base address or address of ar[0] elements, Let’s observe the below code snippet:

int ar[3]={1,2,3};

cout<<ar; //this line will display ar[] base address as array name represents . //base address, very similar to a pointer variable storing address

cout<<\*ar; //in order to see the value stored at the base location you need to . //dereference using ‘\*’

cout<<\*(ar+1); //will show value at ar[1]

>>That means arrays are pass by reference, hence every change made to array elements in passing function will reflected to real array.

>>By default an array can store only single elements at every index location, unless you are creating an array of user defined object type, where each object has multiple data members or 2d array. For e.g. An array of objects.

>>Array of objects:

struct obj{

string name;

int age;

};

void main(){

obj arr[3] ={{“Arun”,4},{“Rahul”,5},{“Rohit”,6}}; //obj type array declaration

}

Inputting array:

>>You can use simple cin,cout for character arrays just like normal variable:

void main()

{ char ar[5];

cin>>ar;

cout<<ar;

}

>>Assigning elements to an array requires using array indexes otherwise you can use {} during declaration. Just these two method works. The below code doesn’t work:

void main()

{ int ar[2];

ar={1,2};

}

This works:

void main()

{ int ar[2]={1,2};

}

2D Matrix:

>>These array will have 2 indexs, I and j for rows and columns and size=i\*j:

Int ar[i][j];

>>Input: int ar[2][2]={1,2,3,4} or int ar[2][2]={{1,2},{3,4}}

>> Hence ar[2][2] could be also be seen as a set of 2 arrays of size 2.

**Strings:**

>>Character of arrays or string variable

>>Valid Declarations:

char str[n], const char \*str, or string str

>>Input:

1. char str[]=”arun”,const char \*str=”arun”; or string str=”arun”;
2. char str[4];

for(i=0;i<4;i++)

{ cin>>str[i];}

cin>>str;

cin.getline(str,4);

fgets(str,4,stdin);

>>Output: cout<<str or puts(str)

>>Iteration: both char str[], char \*str, string str can be iterated using index

>>Useful functions for char str[] or char \*str:

1. strlen(str)- counts the no of characters in str[]
2. strcat(str1,str2)- concates str2 with str1 and puts in str1
3. strcmp(str1,str2)- compares str1 with str2 and returns 0 if both are equal
4. strcpy(str1)
5. toupper()/tolower()

>>Functions for string variable:

1. s.length()
2. s1.find(s2)-fill search s2 in s1 and return the result to a size\_t var
3. s.substr(1,2)- will return a substring of size 2 starting from 1 index in s
4. tolower()/toupper()-works for single string elements in each index
5. s1.replace(i,s2\_size,s2)

>>char s[], char \*s are pretty much same, whatever works for s[] works for \*s too, except you can’t assign like s=”Arun”; const char \*s=”arun” works

>>char \*s=”arun”; //doesn’t work

const char \*s=”arun”; //works

char s[]=”arun”;

char \*c=s;

cout<<c[0]; //op:a

cout <<c; //op:arun

char c[10]="arun";

char \*d;

strcpy(d,c); //doesn’t work, d need space to be allocated to it first

cout<<d;

const char \*d="arun";

char c[10];

strcpy(c,d); //this works

cout<<c;

const char \*a="varun";

const char \*b="tarun";

string s=a; // works

string s=a+b; //doesn’t work cuz + can’t be used to add 2 const char \*

cout<<s;

>>char \*- + operator can’t be used to append 2 char \* or const char \*

Direct assignment can’t be done unless declared const

Strcpy(char \*,char \*), strcpy(char \*,char []) doesn’t work

Strcpy(char[],char \*) works

String = char[]+char[], string=char[]+char \* works

String =const char\* works

String=char \*+char \* doesn’t work

string.replace(I,size,const char \*) works

string.replace(I,size,char []) works

const char\*=const char \* works

char []=char [] doesn’t work

int main()

{

char s[10]="arun";

string s1="Va";

char s2[7]="Biswas";

const char \*s3="Biswas";

s1.replace(1,4,s3);

string s4=s;

char s5[s1.length()];

strcpy(s5,s1.data());

cout<<s5;

return 0;

}

**Loops:**

* **For Loop-**

Traditional way

For(int i=0;i<n;i++)

{cout<<a

}

For(i=0,j=0;i<n,j<n;i++,j++)

{

}

**New way: For each Loop**

Int a[],I;

For(int i:a){ //i instead of pointer will used as a variable to store a[] elements

cout<<i; }

for(auto i:a)

{cout<<a;}

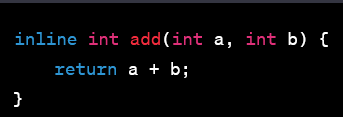
**Functions:**

* A Function has 3 main parts:

1. **Name, return type**
2. **Arguments**
3. **Body**

* **Types of functions:**

1. **Normal function**
2. **Recursive**
3. **Inline-** Inline functions can be declared using following syntax, all you need to do is add **inline** keyword in front of function name.

****

Whenever an inline function call is encountered by the compiler it will replaced by function code. This improve performance only in cases where calling and transferring cpu control to function takes more time than function execution hence only for small function.

1. **Constant-** Constant function can’t change non static data members of an class
2. **Final-** Final functions cannot be inherited and overloaded

* **Default argument values.**

If you are writing a function with arguments, then you can also set some default values for these arguments. Which will be used in case the argument value is not provided during the function call.

**STL:**

>>A collection of containers/data structures with dynamic sizing and bunch of inbuilt useful functions.

**Vector:**

>>Vector: by default vector is LIFO stack when using functions like push\_back() and pop\_back()

>>Declaration:

#include<vector>

void main(){

vector<int> ar; //size not needed to be mentioned

}

>>Iterator: STLs come with the feature of iterators, it’s just a pointer used to traverse vector(STLs) elements.

#include<vector>

#include<iterator>

void main()

{ vector<int> ar;

vector<int>::iterator itr;

for(itr=ar.begin();itr!=ar.end();itr++) //itr is a pointer it will store address of { ar.begin()/ar[0]

cout<<\*itr; //\* must be used to dereference

the pointer to get value

}

}

>>Auto iterator: Does the same job of iterator but no additional header file or use scope resolution operator :: is required. Just declare like:

#include<vector>

void main()

{ vector<int> ar;

for( auto itr=ar.begin();itr!=ar.end();itr++) //itr is a pointer it will store address of { ar.begin()/ar[0]

cout<<\*itr; //\* must be used to dereference

the pointer to get value

}

}

**Lists:**

>>Same as vector except it has functions like, push\_front(), pop\_front()

As such insertion and deletion can take place at both front and rear.

**Deque:**

>>Deque is the best sequential STL to use. It has features of vector and lists.

>>One more additions over those is that deque also allows to delete from in between

**>>Functions:**

1. v.begin()- returns address of first vector element, used for iterators
2. v.end()- returns address of last vector element, used for iterators
3. v.at(i)- return the element in v at index i
4. v.size()- returns the size of vector v
5. v.push\_back(x)- add element x at the end of vector v
6. v.pop\_back()- removes elements
7. v.erase(itr)- deletes the element pointed at by the iterator. Only usable in deque, maps, multimaps
8. advance(itr2, 3);
9. v.clear()- empties the stl
10. v.remove(‘’)-removes the specified element

**Array(STL):**

1. a.fill(value)- fills all array positions with value
2. a1.swap(a2)- swaps contents of a1 and a2
3. a.empty()- empties a

>>Declaration-

Void main()

{ array<int,size> ar={1,2,3,4};

}

**Maps:**

>>Declaration-

#include<map>

Void main(){

Map<int,int> m;

}

>>Insertion-

m.insert({a,b}); or m[a]=b;

>>Retrieving value-

m.at(a)=b or m[a]=b or m.find(a)=b

m.erase(a)= a is index value, in vector a will be itr

**Multimap in stl:**

Note:-

Multimap doesn’t support the [] operator as such you can use

Multimap[key]=data; // for insertion

Or

Cout<<Multimap[key]; //for output like you can in map

**OOPS:**

>>Functions having the return type of a class will return the object of class

Class A{

Int x;

Int y;

A func(int a,int b){

A obj;

Obj.x=a;

Obj.y=b;

Return obj;

}

};

>>**Virtual Function-** Virtual functions are used to solve static binding problem. This problem occurs in class pointer objects.

Class A{

Disp(){ cout<<”Disp of class A”;}

};

Class B:public A{

Disp(){ cout<<”Disp of class B”;} //overriding Disp() of A in B

};

Void main(){

A \*obj1;

B obj2;

Obj1=&obj2;

Obj1->Disp(); //Trying to class Disp() of B using pointer object of A

}

**>>Pure virtual function:** virtual int sum()=0;

**>>Abstract class:** Any class having atleast one pure virtual function is considered an abstract class. You can’t make object of abstract class. In broader terms even base class like animal which describes some base properties for no of different classes can be called an abstract class too.

>>2 ways to create class objects:

1. class a{

};

void main(){

a obj; //this line also invoke a constructor

}

1. class a{ };

void main(){

a \*obj;

obj= new a(); // this line allocates data space of type ‘a’ to obj

}

**>>Constructors**- A constructor is function with same name as class and no return type. For e.g.

class A{

A(){

Cout<<”Constructor of A”;

}

};

Constructor is auto invoked when declaring a class object within a function. For e.g. for the above class A if we create create an object of A in void main

Void main(){

A a; // this line is just not object declaration, but also invokes the default constructor of A, and will o/p “Constructor of A”

}

Constructor function will allocate memory and create the object of A. Also constructor function for every object is only called once during object initialization, it can’t be invoked explicitly

There are 3 types of constructor:

1. Default- You don’t write any body for this function explicitly

class a{

};

void main(){

a obj; //this line also invoke a default constructor of a

}

* A default constructor can also work as a setter method, if the the data members are public and you need to initiate the value for these data member only once during object creation. This feature is available in only C++ 20

#include <iostream>

using namespace std;

class A{

public:

int a;

int b;

};

int main()

{

A a(1,2);

cout<<a.a;

cout<<a.b;

return 0;

}

1. Parameterized
2. Copy

>>In case of multilevel inheritance the object of derived class will access overridden function

>>In case of multilevel inheritance constructors are invoked starting with lower most base class

**Access Specifiers:**

1. **Private-** Can’t be inherited or accessed directly from outside of class, except friend class and friend functions. This the default access specifiers
2. **Public-** Can be inherited as well accessed from outside class using class object.
3. **Protected-** Lies between **private** and **public** as it can be inherited but can’t be accessed from outside class

Note:- Friend functions can access private and public members both of the class

**>>Inheritance modes:**

1. **Private-** This also the default inheritance mode for classes in cpp. Under this mode inherited members (public and protected) become private in derived class.
2. **Public-** Under this mode inherited members (public and protected) stay the same in derived class too, public will be public, protected will be protected.
3. **Protected-** Under this mode inherited members (public and protected) become protected in derived class.

**Templates:**

>>A very useful cpp feature, allows you to create generic functions.

>>Generic functions- Function which can work for a variety of different type of arguments.

>>So usually where you will use polymorphism/function overloading to create multiuse functions for different arquement types, you can do the same using template functions.

Code Snippet:

template<typename T>T add(T x,T y) //T is the generic data type here

{ //<class T> instead of <typename T> works too

return(x+y);

}

void main()

{

cout<<add(1,2); //

cout<<add(1.2,2.2);

cout<<add(‘a’,’b’);

}

>>There are also **template classes** which are used to create dynamic template type data structure. For e.g. You want to create array of dynamic data type, or class for different data type objects.

**File handling:**

>>There are 5 functions primary used for file handling- open(), close(), create(), read(), write().

>>Thes functions are part of fcntl.h> header file.

>>open() & create function return FD on success which can be further used by read(), write() functions in argument to perform on a particular file.

**Scope and Namespace:**

>>Every cpp variable function exists within the scope of another class or function.

>>A classic example to understand this is that you can’t use simple cin, cout in some compilers, instead you have to use std::cin, std::cout. Even after including <iostream.h> in the program.

>>So it why is it so? There is another 2 usages of :: scope resolution operator. That is you can use it to define functions outside class.

**Some good practices:**

>>Some time instead of operating directly using temp variable helps.

>>Don’t enclose in () bracket after return.

**Very Important Codes:**

1. Convert integer to string:

#include<sstream.h>

void main()

{ int n=5;

stringstream ss;

ss<<n;

string s;

s=ss.str();

}

Or simple s=to\_string(n) works too

2. Convert string to char array

String s;

char str=new char(s.length);

ctrcpy(str,s.data());

3. Convert integer to string;

#include<ctype>

Void main(){

String s;

Int n;

Cin>>n;

}

* **Denomination problem:**

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

int amt=23;

int ar[3]={1,2,5};

int j;

for(int i=2;i>=0;i--){

if(ar[i]<amt||ar[i]==amt)

{ j=trunc(amt/ar[i]);

cout<<j<<" ";

amt=amt-(j\*ar[i]);

}

else{

i--;

}

}

return 0;

}

* **12 hr time to 24 hr format:**

#include <bits/stdc++.h>

#include <ctype.h>

#include <sstream>

using namespace std;

string timeConversion(string s) {

int size=s.length();

string s1;

int n;

char \*str=new char[size];

strcpy(str,s.substr(0,2).data());

n=atoi(str);

if(s[size-2]=='P' && n<12){

   n=12+n;

   s1=to\_string(n);

   s.replace(0,2,s1);

}

else if(s[size-2]=='P' && n==12){

   s1=to\_string(n);

   s.replace(0,2,s1);

}

else{if(n==12){

    s[0]='0';

    s[1]='0';

}

}

s[size-1]=' ';

s[size-2]=' ';

return(s);

}

int main()

{

    ofstream fout(getenv("OUTPUT\_PATH"));

    string s;

    getline(cin, s);

    string result = timeConversion(s);

    fout << result << "\n";

    fout.close();

    return 0;

}

* **Finding substring using find():**

#include <string.h>

#include <string>

using namespace std;

int main()

{

string s1="arun";

string s2="varun";

size\_t pos=s2.find(s1);

if(pos!=string::npos){

cout<<"substring found";

}

return 0;

}

* **Using pointers for array:**

int main()

{

char c[10]="arun";

char \*d=c;

cout<<d; //op:arun

return 0;

}

This only works for char array, in case of integer array dereferencing is needed

* **Cyclic sum:**

#include <bits/stdc++.h>

using namespace std;

void findTheSingleDigit(int n) {

   if (n == 0) {

      cout << 0;

   }

   else if (n % 9 == 0) {

      cout << 9 << endl;

   }

   else {

      cout << n % 9 << endl;

   }

}

int main() {

   int n = 4543;

   findTheSingleDigit(n);

   return 0;

}

#include <iostream>

#include <map>

using namespace std;

int main()

{

string s;

cout<<"Enter the string\n";

cin>>s;

multimap<int,char> typeList;

int a,b,c,d,e,f;

for(int i=0;i<s.length();i++){

if(s[i]=='('){

a++;

typeList.insert({1,'o'});

}

else if(s[i]=='{'){

b++;

typeList.insert({2,'o'});

}

else if(s[i]=='['){

c++;

typeList.insert({3,'o'});

}

else if(s[i]==')'){

d++;

typeList.insert({1,'c'});

}

else if(s[i]=='}'){

e++;

typeList.insert({2,'c'});

}

else if(s[i]==']'){

f++;

typeList.insert({3,'c'});

}

else{

cout<<"Invalid statement";

return 1;

}

}

for(auto itr=typelist.begin();typeList[i]!=typeList.back();i++){

if(typeList[i]=='o'&&typeList[i+1]=='c') {

cout<<"Invalid statement";

return 1;

}

}

if((a==d)&&(b==e)&&(c==f)){

cout<<"Valid statement";

}

return 0;

}

\*Design a class named Box whose dimensions are integers and private to the class. The dimensions are labelled: length l, breadth b, and height h.

The default constructor of the class should initialize l, b, and h to 0.

The parameterized constructor Box(int length, int breadth, int height) should initialize Box's l,b and h to length, breadth and height.

The copy constructor Box(Box B) should set l, b and h to B's l, b and h, respectively.

Apart from the above, the class should have 4 functions:

1. int getLength() - Return box's length
2. int getBreadth() - Return box's breadth
3. int getHeight() - Return box's height
4. long long CalculateVolume() - Return the volume of the box

Overload the operator < for the class Box. Box A < Box B if:

1. A.l < B.l
2. A.b < B.b and A.l == B.l
3. A.h < B.h and A.b == B.b and A.l == B.l

Overload operator << for the class Box().

If B is an object of class Box:

cout << B should print B.l , B.b and B.h on a single line separated by spaces.

class Box{

    int l,b,h;

    public:

    Box(){

      l=0,b=0,h=0;

    }

    Box(int length,int breadth,int height){

    l=length;

    b=breadth;

    h=height;

    }

    Box(Box &B){

        l=B.l;

        b=B.b;

        h=B.h;

    }

    int getLength() // Return box's length

    {

        return(l);

    }

    int getBreadth () // Return box's breadth

    {

        return(b);

    }

    int getHeight () //Return box's height

    {

        return(h);

    }

    long long CalculateVolume(){

        return(l\*b\*h);

    }

    bool operator<(Box &c1){

        if ((l<c1.l)||((b<c1.b)&&(l==c1.l))||((h<c1.h)&&(b==c1.b)&&(l==c1.l))){

         return true;

        }

        else{

            return false;

        }

    }

    friend ostream& operator<<(ostream& os, const Box& dt);

};

ostream& operator<<(ostream& os, const Box& dt)

{

    os << dt.l << ' ' << dt.b<< ' ' << dt.h;

    return os;

}

//Operator Overloading

#include<iostream>

using namespace std;

class Complex

{

public:

    int a,b;

    void input(string s)

    {

        int v1=0;

        int i=0;

        while(s[i]!='+')

        {

            v1=v1\*10+s[i]-'0';

            i++;

        }

        while(s[i]==' ' || s[i]=='+'||s[i]=='i')

        {

            i++;

        }

        int v2=0;

        while(i<s.length())

        {

            v2=v2\*10+s[i]-'0';

            i++;

        }

        a=v1;

        b=v2;

    }

};

//Overload operators + and << for the class complex

//+ should add two complex numbers as (a+ib) + (c+id) = (a+c) + i(b+d)

Complex operator+(Complex A,Complex B){

     Complex C;

     C.a=A.a+B.a;

     C.b=A.b+B.b;

     return C;

     }

//<< should print a complex number in the format "a+ib"

ostream& operator<<(ostream& os, Complex C)

{

    os << C.a <<"+i"<< C.b;

    return os;

}