Lecture - 1

1. C++ —Bjarne stroustrup , 1979, as a extension of c.
2. Install vs code- source code editor
3. Intall g++ — c++ compiler
4. Intall mingw – basically this installs g++ gnu compiler
5. Install c/c++ extension in vs code and code runner extension

//what is gnu?

Lecture – 2

1. C++ - fast program , more control over system resources and memory management, with high performance.
2. C++ major updates – 2011-c++11, 2014-c++14, 2017-c++17
3. #include<iostream>
4. int main(){
5. std::cout<<"hello world";
6. return(0);
7. }
8. #include<iostream> - Header file (“iostream” functions for input outputs and more.)
9. Int – integer, main – function

Lecture – 3

1. Variables – containers that holds data
2. Comments – ( // ) used for remembering or providing information for code , compiler ignores this , types- single line comments and multiline comments(/\* comments \*/).

Lecture - 4

1. Types of variables:-
   1. Int- integers
   2. Float-decimal values
   3. Char- stores characters
   4. Double- float but bigger (decimal no. of high precision)
   5. Boolean -true, false
2. Syntax :- data\_type variable\_name =value
3. Types of variables based on scope(region of code where the existence of variables is valid):-
   1. Local variables
   2. Global variables
4. Data types(type of data a variable can hold)
   1. Built in data type- mentioned above
   2. User defined data types
   3. Derived data types
5. User defined data type:-
   1. Struct(structure)
   2. Union
   3. Enum
6. Derived data types:-
   1. Array
   2. Function
   3. Pointer
7. Rules for declaring variables:-
   1. All variables must start with a alphabet or underscore
   2. Variable names are case sensitive
   3. No spaces or special characters are allowed
   4. Can’t use c++ reserved key word

Lecture-5

Basic input/output

1. C++ comes with libraries which helps us in performing input output. In c++ sequence of bytes corresponding to input and output are commonly known as streams.
2. Input stream:- direction of flow of bytes takes place from input device to main memory.
3. Output stream:- direction of flow of bytes take place from main memory to the output device.
4. << this is called insertion operator.
5. >> this is extraction operator.
6. Reserved keywords- they are used by the language itself.

Lecture - 6

Header file and operators

1. There are two types of header files.
   1. System header files – it comes with the compiler
   2. User defined header files - it is written by the programmer (includes a file (which should be available in your current directory.))
2. Operators in c++
   1. Arithmetic operators
      1. +
      2. –
      3. \*
      4. /
      5. %-modulus
      6. A++ - increment – first print then increment
      7. A-- \_ decrement – first print then decremnet
      8. ++a – first increment then decrement
      9. –a – first decrement then increment
   2. Assignment operators
      1. =
   3. Comparision operators
      1. ==
      2. <
      3. >
      4. <=
      5. >=
      6. !=
   4. Logical operators
      1. && - and operator
      2. || - or operator
      3. ! – not operator
   5. Bit wise operator-
   6. Scope resolution operator
      1. ::

Lecture - 7

#include<iostream>

using namespace std;

int c = 45;

int main(){

    // \*\*\*\*\*\*\*\*\*\*\*\*\*Build in Data types\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    // int a, b, c;

    // cout<<"Enter the value of a:"<<endl;

    // cin>>a;

    // cout<<"Enter the value of b:"<<endl;

    // cin>>b;

    // c = a + b;

    // cout<<"The sum is "<<c<<endl;

    // cout<<"The global c is "<<::c;

    // \*\*\*\*\*\*\*\*\*\*\*\*\* Float, double and long double Literals\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    // float d=34.4F;

    // long double e = 34.4L;

    // cout<<"The size of 34.4 is "<<sizeof(34.4)<<endl;

    // cout<<"The size of 34.4f is "<<sizeof(34.4f)<<endl;

    // cout<<"The size of 34.4F is "<<sizeof(34.4F)<<endl;

    // cout<<"The size of 34.4l is "<<sizeof(34.4l)<<endl;

    // cout<<"The size of 34.4L is "<<sizeof(34.4L)<<endl;

    // cout<<"The value of d is "<<d<<endl<<"The value of e is "<<e;

    // \*\*\*\*\*\*\*\*\*\*\*\*\*Reference Variables\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    // Rohan Das----> Monty -----> Rohu ------> Dangerous Coder

    // float x = 455;

    // float & y = x;

    // cout<<x<<endl;

    // cout<<y<<endl;

    // \*\*\*\*\*\*\*\*\*\*\*\*\*Typecasting\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    int a = 45;

    float b = 45.46;

    cout<<"The value of a is "<<(float)a<<endl;

    cout<<"The value of a is "<<float(a)<<endl;

    cout<<"The value of b is "<<(int)b<<endl;

    cout<<"The value of b is "<<int(b)<<endl;

    int c = int(b);

    cout<<"The expression is "<<a + b<<endl;

    cout<<"The expression is "<<a + int(b)<<endl;

    cout<<"The expression is "<<a + (int)b<<endl;

    return 0;

}

Type casting – it is used to change one data type to another.

Lecture - 8

Constants, manipulators & operator precedence

#include<iostream>

#include<iomanip>

using namespace std;

int main(){

    const int a=35;

    cout<<"for constant value of a"<<a<<endl;

    cout<<"these are for maipulators";

    int b=546, c=5684 ,d=45;

    cout<<"the value of the variables are"<<endl<<setw(4)<<a<<endl<<setw(4)<<b<<endl<<setw(4)<<c<<endl<<setw(4)<<d<<endl;

    //precedence is which operator will evaluate first and associativity is direction of parsing (left to right or right to left)

    cout<<"operator precidence"<<endl;

    int e=(((c\*a)+b)-d);

    cout<<e;

    return(0);

}

Lecture – 9&10

Control structure, if else, switch case statement

1. Control structure:-
   1. Sequence structure- entry – action 1 - action 2- exit
   2. Selection structure – entry -condition – if true a1 – if false – a2 -exit.(if/else)
   3. Loop structure – entry – loop – condition – true -a1 – false – a2 check condition again if false – exit.
2. If else statement :-
   1. If (i=3){
   2. Cout <<I;
   3. I++;
   4. }
3. If ese ladder
4. Switch case
   1. Switch(expression){
   2. Case1:{
   3. Action
   4. }
   5. Default {
   6. Action 4}
   7. }
5. Loops in c++
   1. For loop
   2. While loop
   3. Do while loop
6. For loop syntax:-
   1. For(initialization; condition; upgradation)
   2. {
   3. Loop body(c++ code)
   4. }
7. While loop syntax:-
   1. While(condition)
   2. {
   3. C++ statements;
   4. }
8. Do while loop syntax:-
   1. Do{
   2. C++ statement;
   3. }While(condition);

Lecture – 11

Break and continue statements: -

#include<iostream>

using namespace std;

int main(){

    // for (int i = 0; i < 40; i++)

    // {

    //     /\* code \*/

    //     if(i==2){

    //         break;

    //break stops the current itteration , and gets out of the loop

    //     }

    //     cout<<i<<endl;

    // }

    for (int i = 0; i < 40; i++)

    {

        /\* code \*/

        if(i==2){

            continue;

            //continue stops the current itteration and jumps on to the next one

        }

        cout<<i<<endl;

    }

    return(0);

}

Lecture - 12

Pointers: -

Pointers is a type of data type which holds the address of other data types.

1. (“\*”) --- dereference operator:- ex.
   1. Int a=3;
   2. Int\* b=&a;
   3. //”&” -- address of operator.
2. Pointers basically stores addresses of other data types.
3. #include<iostream>
4. using namespace std;
5. int main(){
6. int a=3;
7. int\* b=&a;
8. cout<<"address of a\n"<<&a<<endl;
9. cout<<"address of a\n"<<b<<endl;
10. //&--->adress of operator
11. cout<<"the value of address b\n"<<\*b;
12. //\*---->value at address--->derefrence variable
13. return(0);
14. }

Lecture - 13

Array:-

1. It is the collection of items of similar type stored in contiguous memory location.
2. Sometimes, a simple variable is not enough to hold all the data.

Syntax example: - int marks[4] = {23, 24, 25, 26};

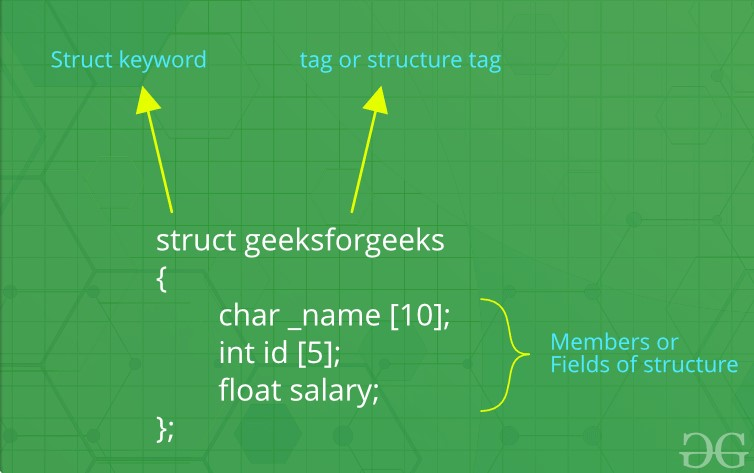
1. #include<iostream>
2. using namespace std;
3. int main(){
4. // Array Example
5. int marks[] = {23, 45, 56, 89};
6. int mathMarks[4];
7. mathMarks[0] = 2278;
8. mathMarks[1] = 738;
9. mathMarks[2] = 378;
10. mathMarks[3] = 578;
11. cout<<"These are math marks"<<endl;
12. cout<<mathMarks[0]<<endl;
13. cout<<mathMarks[1]<<endl;
14. cout<<mathMarks[2]<<endl;
15. cout<<mathMarks[3]<<endl;
16. // You can change the value of an array
17. marks[2] = 455;
18. cout<<"These are marks"<<endl;
19. // cout<<marks[0]<<endl;
20. // cout<<marks[1]<<endl;
21. // cout<<marks[2]<<endl;
22. // cout<<marks[3]<<endl;
23. for (int i = 0; i < 4; i++)
24. {
25. cout<<"The value of marks "<<i<<" is "<<marks[i]<<endl;
26. }
27. // Quick quiz: do the same using while and do-while loops?
28. // Pointers and arrays
29. int\* p = marks;
30. cout<<\*(p++)<<endl;
31. cout<<\*(++p)<<endl;
32. // cout<<"The value of \*p is "<<\*p<<endl;
33. // cout<<"The value of \*(p+1) is "<<\*(p+1)<<endl;
34. // cout<<"The value of \*(p+2) is "<<\*(p+2)<<endl;
35. // cout<<"The value of \*(p+3) is "<<\*(p+3)<<endl;
37. return 0;
38. }

Pointer arithmetic: -

Address new = address current + I \* sizes(data types)

Lecture - 14

Structure, union, enums: -

1. Unlike arrays, structure in c++ are user defined data types which are used to store group of items of non-similar data types.
2. Syntax: - 
3. // typedef: - typedef is a keyword used to assign a new name to a exiting data type.
4. Syntax: -
5. Typedef current\_name new\_name;
6. Example
7. typedef struct employee
8. {
9. int eid[10];
10. float salary;
11. char position;
12. } ep;
13. –
14. –
15. Union: -
16. Unions are similar to structure but they provide better memory management then structure. Unions are shared memory so only 1 variable can be used at a time.
17. Example:-
18. {
19. /\* data \*/
20. int rice; //4
21. char car; //1
22. float pounds; //4
23. };
24. Enums: -
25. Enums are user defined types which consists of named constant. enums are used to make program more readable.
26. Example:-
27. int main(){
28. enum Meal{ breakfast, lunch, dinner};
29. Meal m1 = lunch;
30. cout<<m1;
31. return 0;
32. }

Lecture - 15

Function and function prototypes: -

1. Function is the main part of top-dwn structured programming. we break the code in small pieces and make functions of that code. functions help us to reuse the code easily.
2. Example:-
3. int sum(int a, int b){
4. int c = a+b;
5. return c;
6. }
7. FUNCTION PROTOTYPE: -
8. The function prototype is the template of function which tells the details of the function to the compiler. function prototype helps us to define a function after the function call.
9. Syntax / example:-
   1. int sum(int a, int b); //Acceptable
   2. int sum(int a, b); // Not Acceptable
   3. int sum(int, int); //Acceptable
10. type function-name (arguments);

Lecture -16

Call by value and call by refrence:-

#include<iostream>

using namespace std;

//call by refrence using pointers

//void swappointer(int \*a, int \*b){

//    int temp=\*a;

//    \*a=\*b;

//    \*b=temp;

//}

//call by refrence using c++ refrence variable

void swaprefrencevar(int &a, int &b){

    int temp=a;

    a=b;

    b=temp;

}

int main(){

    int x=4, y=5;

    cout<<"this is the value of x"<<x<<"this is the value of y"<<y<<endl;

    //swappointer(&x , &y);

    //this will swap a and b using pointers

    //cout<<"this is the value of x after swap"<<x<<"this is the value of y apfter swap"<<y<<endl;

    //this will swap a and b using refrence variable

    swaprefrencevar(x, y);

    cout<<"this is the value of x by refrence variable"<<x<<"and y"<<y<<endl;

    return (0);

}

Lecture – 17

Inline functions, default arguments, constant argumanet:-

1. inline function:-
2. inline functions are used to reduce the function call.when one function is being called multiple times in the program it increases the execution time , so iline function is used to reduce time and increase program efficiency. If the inline function I being used when the function is called the inline function expands the while function code at the point of a function call, instead of running the function.
3. inline int product(int a, int b){
4. return a\*b;
5. }
6. Default arguments:-
7. Default arguments are those values which are used by the function if we don’t input our values. It is recommended to write default arguments after the other arguments.
8. Constant arguments:-
9. Constant arguments are used when you don’t want your values to change or modified by the function

Lecture - 18

Recursion and recursive functions

1. When a function calls itself, it is called recursion and the function which is calling itself is called a recursive function. The recursive function consists of a base case and recursive condition. It is very important to add a base case in recursive function otherwise recursive function will never stop executing.

Lecture - 19

Function overloading: -

1. Function overloading is the process to make more then one variable with the same name but different parameters, numbers, or sequence.
2. Example:-
3. #include<iostream>
4. using namespace std;
5. int sum(float a, int b){
6. cout<<"Using function with 2 arguments"<<endl;
7. return a+b;
8. }
9. int sum(int a, int b, int c){
10. cout<<"Using function with 3 arguments"<<endl;
11. return a+b+c;
12. }
13. // Calculate the volume of a cylinder
14. int volume(double r, int h){
15. return(3.14 \* r \*r \*h);
16. }
17. // Calculate the volume of a cube
18. int volume(int a){
19. return (a \* a \* a);
20. }
21. // Rectangular box
22. int volume (int l, int b, int h){
23. return (l\*b\*h);
24. }
25. int main(){
26. cout<<"The sum of 3 and 6 is "<<sum(3,6)<<endl;
27. cout<<"The sum of 3, 7 and 6 is "<<sum(3, 7, 6)<<endl;
28. cout<<"The volume of cuboid of 3, 7 and 6 is "<<volume(3, 7, 6)<<endl;
29. cout<<"The volume of cylinder of radius 3 and height 6 is "<<volume(3, 6)<<endl;
30. cout<<"The volume of cube of side 3 is "<<volume(3)<<endl;
31. return 0;
32. }

Lecture - 20

1. Object oriented programming: -
   1. It works on the concept of classes and objects
   2. A class is a template to create objects
   3. It treats data as a critical element
   4. Decomposes the problem into objects and builds data and objects around the objects.
2. Classes: - basic templates for creating objects
3. Objects: - basic run time entities
4. Data abstraction and encapsulation : - wrapping data and function into single unit
5. Inheritance: - properties of one class can be inherited into others
6. Polymorphism: -ability to make more the one forms
7. Dynamic binding: - code which will execute is not known until program runs.
8. Message passing – objects.message(information) call format.
9. Benefits of using oops:-
   1. Better code reusability using objects and inheritance
   2. Principle of data hiding helps build secure system
   3. Multiple objects can co-exist without any interference
   4. Software complexity can be easily managed

Lecture: - 21

Classes, public & private class modifiers in c++:-

1. Classe: - classes are user defined data types and are a template for creating objects. Classes consists of variables and function which are also known as class variables.
2. Public access modifiers:- all the variables and functions declared under the public access modifier will be available for everyone.they can be accesed both inside and outside the class.
3. Private access modifiers:-all the variables and functions declared under the private modifier can only be used inside the class.they are not permissible to be used by any object or function outside the class.

Lecture : - 22

1. Oops: - classes and objects
2. C++  initially called  c wth classes by stroustroup
3. Class extension of structure (in c)
4. Structures had limitations
   1. members are public
   2. no methods
5. Classes = structure + more
6. Classes can have methods and properties
7. Classes can make a few members as private and few as public
8. Structure in c++ are by default typedef
9. You can declare objects along with the class declaration like this
   1. Class employee{
      1. Class definition
   2. }harry , rohan , lovish;
10. Harry.salary = 9 makes no sense if the salary is private
12. Nesting of member function
13. if one member function Is called inside other member function of the same class it is called nesting of the member function.

Lecture - 23

Memory allocation and using array in classes: -

1. The way memory is allocated to variables and functions of the class is different even though they both are from the same class.
2. Th memory is only allocated to the variables of the class when the object is created. The memory is not allocated to the variable when the class is declared. At the same time single variables can have different values for different objects so every object has an individual copy of all the variables of the class. But the memory is allocated to the function only once when the class is declared. So the object don’t have individual copies of function only one copy is shared among each object.
3. Array in class:-arrays are used to store multiple values of the same type. An array is very helpful when multiple variables are required, instead of making multiple variables one array can be used to store multiple values. Array store data in sequential order.

Lecture - 24

Static data member and methods in c++ : -

1. Static data member: - when a static data member is created there is only a single copy of data member which is shared between all the objects of the class.
2. Static method : - when a static member is created they become independent of any object and class. Static methods can only access static data members and static methods. Static methods can only be accessed using the scope resolution operator.

Lecture – 25

Array in objects and passing objects as function arguments: -

1. An array of objects is declared as same as any other data type array. An array of objects consists of class objects as its element. If the array consists of the class objects it is called an array of objects.
2. int main()
3. {
4. Employee fb[4];
5. for (int i = 0; i < 4; i++)
6. {
7. fb[i].setId();
8. fb[i].getId();
9. }
10. return 0;
11. }
12. Passing objects as function arguments: - objects can be passed as function arguments. This is useful when we want to assign the values of a object to current object.
13. int main(){
14. complex c1, c2, c3;
15. c1.setData(1, 2);
16. c1.printNumber();
17. c2.setData(3, 4);
18. c2.printNumber();
19. c3.setDataBySum(c1, c2);
20. c3.printNumber();
21. return 0;
22. }

Lecture = 26

1. Friend function: -
2. Friend function are those functions that have the right to access the private data members of class even though they are not defined inside the class. It is necessary to write the prototype of the friend function. One main thing to note here is that to write the prototype of the friend function in the class it will not make that function a member of the class.
3. Properties of friend function:-
   1. Not in the scope of class
   2. Since it is not in the scope of class it can’t be called from he object of the class
   3. Can be invoked without the help of any object
   4. Usually contains the objects as arguments
   5. Can be declared inside the public or private section of the class
   6. It cannot access the members directly by their names and need objects\_name.member\_name to access any member

Lecture - 27

Friend class: -

1. Member friend function in c++ : -
   1. Friend function are those functions that have the access to private members of the class in which they are declared. The main thing to note here is that only that function can access the member function which is made a friend of the other class.
2. Friend class: -
   1. Friend are those classes that have the permission to access the private members of the class in which they are declared.the main thing to note here is that if the class is made friend of another class then it can access all the private members of the class

Lecture - 28

1. #include<iostream>
2. using namespace std;
3. class y;
4. class x{
5. int data;
6. public:
7. void setvalue(int value){
8. data = value;
9. }
10. friend void sum(x ,y);
11. };
12. class y{
13. int num;
14. public:
15. void setvalue(int value){
16. num=value;
17. }
18. friend void sum(x , y);
19. };
20. void sum(x o1, y o2){
21. cout<<"the sum of object o1 and o2 is "<< o1.data + o2.num;
22. }
23. int main(){
24. x a1;
25. a1.setvalue(30);
26. y b1;
27. b1.setvalue(425);
28. sum(a1, b1);
29. return(0);
30. }

Lecture - 29

Constructors: -

1. A constructor is a special member function with the same name as the class. the constructor doesn’t have a return type. Constructors are used to initialize the objects of its class. Constructors are automatically invoked whenever a object is created.
2. Important characteristics of constructors in c++
   1. A constructor should be declared in the public section of the class
   2. They are automatically invoked whenever a object is created
   3. They cannot return values and do not have return types
   4. It can have default arguments
   5. We cannot refer to their address

Lecture - 30

Parameterized constructor: - these constructors are those constructors that take one or more parameters. default constructors are those constructors that take no parameters. The main thing to note here are that constructors are written in public section of the class and the constructors don’t have a return type.

Lecture – 31

Constructor overloading: - constructor overloading is a concept in which one class can have multiple constructors with different parameters. The main thing to note here is that the constructors will run according to the arguments.

Lecture - 32

Default arguments: - default arguments of a constructor are those which are provided in the constructor declaration. If the values are not provided then calling the constructor use the default arguments automatically.

Lecture - 33

The dynamic initialization of the object means that the object is initialized at the runtime. Dynamic initialization of the object using a constructor is beneficial when the data Is of different formats.

Lecture - 34

Copy constructor: -a copy constructor is a type of constructor that creates a copy of another object, if we want one object to resemble another object we can use a copy constructor . if no copy constructor is written is the program compiler will supply its own copy constructor.

Lecture – 35

Destructor: - a destructor is a type of function which is a called when the object is destroyed. Destructor never takes an argument nor does it return any value.

Lecture – 36

Inheritance in c++ : -

* Reusability is a very important feature of OOP’s.
* In c++ we can reuse a cclass and add additional features to ir
* Reusing classes saves time and money
* Reusing already tested and debugged classes will save a lot of effort of developing and debugging the same thing again

What is inheritance in c++: -

* The concept of reusability in c++ is supported using inheritance.
* We can reuse the property of an existing class by inheriting it.
* The existing class is called a base class
* The new class which is inherited from the base class is called a derived class.
* Reusing classes save time and money
* There are different types of inheritance in c++

Forms of inheritance in c++

* Single inheritance
* Multiple inheritance
* Hierarchical inheritance
* Multiple inheritance
* Hybrid inheritance

Single inheritance : - single inheritance is a type of inheritance in which a derived class is inherited with only one base class.

Multiple inheritance: - multiple inheritance are a type of inheritance in which one derived class is inherited with more than one base class.

Hierarchical inheritance : - a hierarchical inheritance is a type of inheritance in which several derived classes are inherited from a single base class.

Multilevel inheritance : - multilevel inheritance is a type of inheritance in which one derived class is inherited from one derived class.

Hybrid inheritance : -

Hybrid inheritance: - hybrid inheritance is a combination of multiple inheritance and multilevel inheritance .

Lecture – 37

Inheritance syntax and visibility : -

// Derived Class syntax

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

{

class members/methods/etc...

}

Note: -

* Default visibility mode is private
* Public visibility mode : public members of the base class

Becomes public members of the derived class

* Private visibility mode: public members of the base class becomes private members are never inherited.

Lecture – 38

Single inheritance: - single inheritance is a type of inheritance in which a derived class is inherited with only one base class.

Lecture – 39

Protected access modifiers: - protected access modifiers are similar to the private access modifiers can be accessed in the derived class whereas private access modifiers cannot be accessed in the derived class.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Public Derivation** | **Private Derivation** | **Protected Derivation** |
| **Private members** | Not Inherited | Not Inherited | Not Inherited |
| **Protected members** | Protected | Private | Protected |
| **Public members** | Public | Private | Protected |

As shown in the table,

* If the class is inherited in public mode then it’s private member cannot be inherited in child class.
* If the class is inherited in public mode then it’s protected member are protected can be accessed in child class.
* If the class is inherited in public mode then it’s public members are public and can be accessed inside the child class and outside the class.
* If the class is inherited in private mode then its private members cannot be inherited in child class
* If the class is inherited in private mode then its protected members are private and cannot be accessed in child class.
* If the class is inherited in private mode then its public members are private and cannot be accessed in child class.
* If the class is inherited in protected mode then its private members cannot be inherited in child class.
* If the class is inherited in protected mode then its protected members are protected and can be accessed in child class.
* If the class is inherited in protected mode then its public members are protected and can be accessed in child class.

Lecture – 40

Multilevel inheritance: - multilevel inheritance is a type of inheritance in which one derived class is inherited from another derived class.

*/\*if we are inheriting class b from a and c from b*

*1.a is the base class for b and b is the base class for c*

*2.a-->b-->c is called inheritance path*

*\*/*

Lecture – 41

Multiple inheritance.

Lecture – 42

*/\*create 2 classes*

*1.simple calculator-takes input of 2 numbers and performs +,-,\*,/.*

*2.scientific calculator - simple calculator + and performs any 4 scientif operations of your choice*

*3.and displays the result using another function*

*4.create another class and inherit using these 2 classes (hybrid calculator).*

*5.create an object in hybrid calculator and display the result in of simple and scientic calulator*

*simplecalc-->scientficcalc-->hybridcalc(with show function).*

*\*/*

Lecture – 43

Ambiguity in inheritance can be defined as when one class is derived for two or more base classes then there are chances that the base classes have function with the same name. so it will confuse derived class to choose from similar name function.to solve this ambiguity scope resolution operator is used.

Lecture – 44

Virtual base class in c++: -

The virtual base class is a concept used in multiple inheritance to prevent ambiguity between multiple instances.

. For example: suppose we created a class “A” and two classes “B” and “C”, are being derived from class “A”. But once we create a class “D” which is being derived from class “B” and “C”.

Lecture – 45

Virtual base class example

Lecture – 46

Constructors in derived class: -

* We can use constructors in derived classes in c++
* If the base class constructor doesn’t have any arguments, there is no need for any constructor in the derived class
* But if there are one or more arguments in the base class constructor, derived class need to pass arguments to the base class constructor
* If both base and derived classes have constructors, base class constructor is executed first.

Constructors in multiple inheritance: -

* In multiple inheritance, base classes are constructed in te order in which they appear in the class declaration
* In multiple inheritance the constructor are executed in the order of inheritance.

Special syntax

* C++ supports a special syntax for passing arguments to multiple base classes
* The constructors of the derived class receives all the respective base classes
* The body is called after the constructors is finished executing
* Derived-Constructor (arg1, arg2, arg3….): Base 1-Constructor (arg1,arg2), Base 2-Constructor(arg3,arg4)
* {
* ….
* } Base 1-Constructor (arg1,arg2)

Special case of virtual base class

* The constructor for virtual base classes are invoked before a non-virtual base class
* If there are multiple virtual base class they are invoked in the order declared
* Any non-virtual base class are then constructed before the derived class constructor is executed

Lecture – 48

Constructor in derived class cpp code.

Lecture – 49

Initialization list in constructors: -

The initialization list in constructors is another concept of initialization the data members of the class. The syntax of the initialization list in constructors is: -

Syntax for initialization list in constructor:

constructor (argument-list) : initilization-section

{

assignment + other code;

}

Lecture – 50

Pointers with new and delete keyword/operator

Lecture – 51

Arrow operator in c++

For the pointer to object and the use of the arrow operator: -

As in code-

1. We create a class complex which contains two private data members “real” and “imaginary”.
2. The class “complex” contains two members function “getdata” and “setdata”.
3. The function “setdata” will take two parameters and assign the values of parameters to the private data members “real” and “imaginary .
4. The function “getdata” will print the values of private data members “real” and “imaginary
5. In the main program objects is created dynamically by using the “new” keyword and its address is assigned to the pointer “ptr”
6. The member function “setdata” is called using the pinter “ptr” with the arrow operator “->” and the values “----,----” are passed
7. The member function “getdata” is called using the pointer “ptr” with the arrow operator “->” and it will print the value of data member
8. Array of object iscreated dynamically by using the “new” keyboard and its address is assigned to the pointer “ptr”
9. The member function “setdata” s called using the pointer “ptr” with the arrow operator “->” and the values “1,4” are passed
10. The member function “getdata” is called using the pointer “ptr” with the arrow operator “->” and it will print the values of data members.

Lecture – 52

Array of objects using pointers in c++: -

Array of the object can be defined as an array that’s each element is an object of the class. In this tutorial , we will use the pointer to store the address of an array of objects.

Lecture – 53

This is keyword that is an implicit pointer . this pointer points to the object which calls the member function.

Lecture – 54

Polymorphism

Poly means several and morphism means form. So we can say that polymorphism is something that has several forms or we can say it as one name multiple forms. There are two types of polymorphism:

* Compile time polymorphism
* Run time polymorphism

Compile time polymorphism: -

In compile time polymorphism, it is already known which function will run compile-time polymorphism is also called early binding , which means that you are already bound to the function call and you know that this function is going to run .there are two types of compile-time polymorphism:

1.function overloading: - this is a feature that lets us create more than one function and the function have the same names but their parameters need to be different. If function overloading is done in the program and function calls are made the compiler already knows that which function to execute.

2.operator overloading: - this is a feature that lets us define operator working for some specific tasks. For example, we can overload the operator “+” and define its functionality to add two strings. Operator loading is also an example of compile time polymorphism because the compile already knows at the compile time which operator has to perform the task.

Run-time polymorphism : -

In the run-time polymorphism, the compiler doesn’t know already what will happen at run time. run time polymorphism is also called late binding. the run time polymorphism is considered slow because function calls are decided at the run time. Run time polymorphism can be achieved from the virtual function.

3.virtual function: -

A function that is in the parent class but redefined in the child class iscalled virtual function. ”virtual” keyword is used to declare a virtual function.

Lecture – 55

Pointer in derived class in c++

In c++ we are provided with the functionality to point the pointer to derived class or base class.

1. We created a class “BaseClass” which contains public data member “var\_base” and member function “display”. The member function “display” will print the value of data member “var\_base”
2. We created another class “DerivedClass” which is inheriting “BaseClass” and contains data member “var\_derived” and member function “display”. The member function “display” will print the values of data members “var\_base” and “var\_derived”

Lecture – 56

Virtual function: -

A member function in the base class which is declared using virtual keyword is called virtual function.they can be redefined in the derived class.

The main thing to note here is that if we don’t use the virtual keyword with the function of the base class then beside of the base class then beside of the point that we have pointed our base call pointer to derived class object still the compiler would have called the function of the base class because this is it’s default behaviour as we have seen in the previous lectures

But we have the virtual keyword with the function of the base class to make is virtual function so when the display function is called by using the base class pointer the display function of the derived class will run because the base class pointer is pointing to the derived class object.

Lecture – 57

More on virtual function and it’s rules: -

As we have seen in the previous tutorial that how virtual function are used to implement run-time polymorphism.

Rule for virtual function

1. They cannot be static
2. They are accessed by object pointers
3. Virtual function can be a friend of another class
4. A virtual function in the base class might not be used
5. If the virtual function is defined in a base class there is no necessity of redefining it in the derived class.

Lecture – 58

Abstract base class & pure virtual function

Pure virtual function in c++  
pure virtual function is a function that doesn’t perform any operation and the function is declared by assigning the value 0 to it. Pure virtual functions are declared in abstract classes.

Abstract base class in c++

Abstract base class is a class that has at least one pure virtual function in it’s body. The classes which are inheriting the base class must need to override the virtual function of the abstract class otherwise compiler will throw an error.

To demonstrate the concept of abstract class and pure virtual function an example program is shown below.

class CWH{

protected:

string title;

float rating;

public:

CWH(string s, float r){

title = s;

rating = r;

}

virtual void display()=0;

};

Lecture – 59

The file is a patent of data which is stored in the disk. Anything written inside the file is called a patent, for example: “**#include**” is a patent. The text file is the combination of multiple types of characters, for example, semicolon “;” is a character.

The computer read these characters in the file with the help of the ASCII code. Every character is mapped on some decimal number. For example, ASCII code for the character “A” is “65” which is a decimal number. These decimal numbers are converted into a binary number to make them readable for the computer because the computer can only understand the language of “0” and “1”.

The reason that computers can only understand binary numbers is that a computer is made up of switches and switches only perform two operations “true” or “false”.

**File Input and Output in C++**

The file can be of any type whether it is a file of a C++ program, file of a game, or any other type of file. There are two main operations which can be performed on files

* **Read File**
* **Write File**

Lecture – 60

File I/O in c++: reading amd writing files

These are some useful classes for working with files in c++

* Fstreambase
* Ifstreamderived from fstreambase
* Ofstreamderived from fstreambase

In order to work with file in c++, you will have to open it, primarily, there are 2 ways to open a file:

* Using the constructor
* Using the member function open() of the class

Lecture – 61

Follow these steps to write into empty file :

1. Create a text file in the same directory as that of the program
2. Create a string variable name
3. Create an object using ofstream passing the text file into it. This establishes a connection between your program and the text file
4. Take input from the user using cin into the string.
5. Pass the name string to the object . the string name gets written in the text file.
6. Disconnect the file with the program since we are done writing to it using h.out.close().

Follow these steps below to read from the fle we just wrote into

1. Create a string variable content.
2. Create an object using ifstream passing the text file, into it.this establishes a new connection between your program and the text file.
3. Fill in the string using the object.use getline , to take into input the whole line from the text file
4. Disconnect the file with the program since we are done reading from it using .close() function.

Lecture – 62

Using member function open:

The member function open is used to connect the text file to the c++ program when passed into it.

1. Unlike what we did earlier passing the text file in the object while creating it, we’ll first just declare an object out(any name you wish) of the type of stream and use its open method to open the text file in the program.
2. We’ll pass some string lines to the text file using the out operation
3. We’ll now close the file using the close function. Now closing is explicitly used to make the system know that we are fone with the file and it is always good to use this.

Using the member function eof:

The member function eof(End-of-file) return a Boolean true if the file reaches the end of it and false if not.

Understanding the snippet below:

1. We’ll first declared an object in (any name you wish) of the type ifstream and useits open method similar to what we did above to open the text file in the program.
2. And now, we’ll declare the string variable st to store the content we’ll receive from the text file sample.txt
3. Now since we are not only wanting the first or some two or three strings present in the text file, but the whole of it, and we have no idea of what the length of the file is, we’ll use a while loop.
4. We’ll run the while loop until the file reaches the end. till then a 0 or false
5. We’ll use getline to store the whole line in the string variable st. don’t forget to include the header file <string>.
6. This program now successfully prints the whole content of the text file.

Lecture – 63

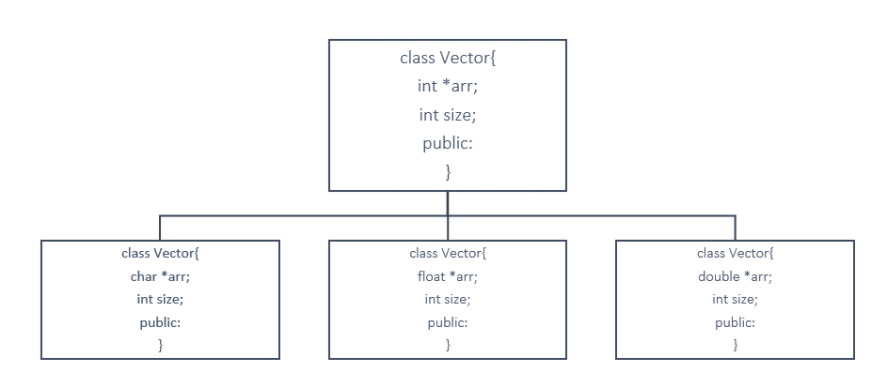
Templates in c++:

What is templates in c++ programming?

A template is believed to escalate the potential of c++ several fold by giving it the ability to define data types as parameters makng it useful to reduce repetition of the same declaration of classes for different data types. Declaring classes for every other data type in the very first place violates the dry(don’t repeat yourself rule of the programming and on the other hand doesn’t completely utilises the potential of c++

It is very analogous to when we said classes are the templates for object, here templates itself are the templates of the classes. That is, what classes are for objects, templates are for classes.

WHY TEMPLATES

1. DRY Rule: -To understand the reason behind using templates, we will have to understand the effort behind declaring classes for different data types. suppose we want to have a vector for each of the three(can be more) data types, int , float and char. then we’ll obviously write the whole thing again and again making it awfully difficult. this is where the saviour comes the templates, it helps parameterizing the data types and declaring it once in the source code suffice. Very similar to what we do in functions.it is because of this also called “parameterised classes”.
2. Generic programming: - it is called generic, because it is sufficient to declare a template once, it becomes general and it works all along for all the data types.
3. 
4. We had to copy the same thing again and again for different data types but a template solves it all. refer the syntax section for how.
5. template <class T>
6. class vector {
7. T \*arr;
8. int size;
9. public:
10. vector(T\* arr)[
11. //code
12. ]
13. //and many other methods
15. };

Lecture-64

*//#include<bits/stdc++.h>*

*#include*<iostream>

using namespace std;

template<class t>*//data type as a variable*

class vector{

    public:

    t\*arr;

    int size;

    vector(int m){

        size=m;

        arr=new t[size];

    }

    t dotproduct(vector &v){

        t d=0;

*for* (int i=0;i<size;i++){

            d+=*this*->arr[i]\*v.arr[i];

        }

*return* d;

    }

};

int main(){

    vector<float>v1(3);

    v1.arr[0]=1.4;

    v1.arr[1]=3.3;

    v1.arr[2]=0.1;

    vector<float>v2(3);

    v2.arr[0]=0.1;

    v2.arr[1]=1.90;

    v2.arr[2]=4.1;

    float a = v1.dotproduct(v2);

    cout<<a<<endl;

*return* (0);

}

Lecture-65

to give a short overview of how templates work with multiple parameters, you can think of it as a function where you have that power to pass different parameters of the same or different data data types. A simple template with two parameters would look something like this. The only effort it demands is the declaration of parameters.

/\*

template<class T1, class T2>

class nameOfClass{

//body

}

\*/

Lecture-66

template parameters may have default arguments. The set of default template arguments over all declaration of a given template.

template<class T, class U = int> class A; template<class T = float, class U> class A; template<class T, class U> class A { public: T x; U y; };

Lecture – 67

function template:-

*#include*<iostream>

using namespace std;

template<class t1,class t2>

float funcaverage(t1 a,t2 b){

    float avg=(a+b)/2;

*return* avg;

}

int main(){

    float a;

    a=funcaverage(5,2);

    cout<<a <<endl;

*return* (0);

}

Lecture-68

#include <iostream>

using namespace std;

void func(int a){

cout<<"I am first func() "<<a<<endl;

}

template<class T>

void func(T a){

cout<<"I am templatised func() "<<a<<endl;

}

Lecture-69

STL:-

An STL is a library of generic function and classes which saves you time and energy which you would have spent constructing for your use. This helps you reuse well tested classes and functions umotenth number of time according to your own convenience.

To put simply, STL is used because it is not a good idea to reinvent something which is already built and can be used to innovate things further. Suppose you go yo a company who builds cars, they will not ask you to start from scratch, but to start from where it is left. This is the basic idea behind using STL.

Component of STL:

we have three components in stl:

1.containers

2.algorithms   
3.iterators

Containers:containers is an object which stores data. We have different containers having their own benefits. These are the implemented templates classes for our use, which can be used just by including this library. You can even customise these templates classes.

Algorithms:

algorithms are a set of instructions which maintains the input data to arrive at some desired result. In stl, we have already written algorithms, for example, to sort some daa structure, or search some element in an array. These algorithms use templates functions.

Iterators:

iterators are objects which refers to an element in a container. And we handle them very much similarly to a pointer. Their basic job is to connect algorithms to the container and play a vital role in manipulation of the data .

Lecture-70

**three components of stl**

**container,** objects which store data

**algorithms,** set of instructions that process data

**iterators,** objects which points to some element in the container

containers are themselves of three types:-

1.sequence containers

2.associative containers

3.derived containers

# sequence containers:-

a sequence container stores the data in a linear fashion. Sequence container includes vector,list(linked list), dequeue etc.

#associate containers:-

an associate container is designed in such a way that enhances the accessing of some elements in the container . It is used when a user wants to fastly reach element. Some of these containers are set,multiset, map, multimap, etc.

#derived containers:-

these containers are derived from either the sequence or associative containers. They often provide you with better methods to deal with your data. They deal with real lifde modelling. Some example of derived containers are stack, queue, priority queue etc.

Lecture – 71

Vector:-

Syntax: - vector<data type> vector \_name

* Push\_back() and size(): -
  + Vectors have a method, push\_back(),to insert element in it from the rear end.
  + We’ll define a variable, size, to define the size of the vector.
  + We’ll then run a loop of size length, to recive the user input and push them back in the vector.
  + We have another method size() which returns the size of the vector .
* Pop\_back(): -
  + This method of the vector deletes the last element sof the vector.

Lecture – 72

A list is a bi directional linear storage of elements. Few key features as to why a list should be used is,

1.it gives faster insertion and deletion operation.

2.it’s access to random elements is slow.

What makes a list different form array?

An array stores the element in a contiguous manner in which inserting some element in the somewhere between calls for a shift in other elements, which is time taking. But in al list we ca simply change the address the pointers pointing to.

Lecture – 73

Map: -

A map in c++ stl is an associative container which stores key value pair. To elaborate a map stores akey of some data type and its corresponding values of some data type. For example: a teacher wants to store the marks of the student which is in future can be accessed byt their names. Here, keys are the students names and their marks are the corresponding values.

Lecture – 74

Functional object: -

A functional object is a function as an object. A function object is a function wrapped in a class so that it is available as an object.