**C ++**

Day 1 and 2 :

C++ is a optional oriented programming and provide functionality that violates the OOP concepts

Source Code

Preprocessor : This helps to remove the comments and include the .h files and expand the headers

Compiler :

lexical analyzer : Convert the word to extension using the tokenizer

Parser : this check whether we have followed the rules / grammer

Code generator : Generate the assembly language convert high level lang to

low level language

Optimizer : It analyzes high-level language code (such as C++, Java, or Python)

and applies various transformations to produce optimized assembly machine code.

Assembler : An assembler is a low-level programming tool that translates

human-readable assembly language code into machine code

Error Collector : Collect all error and through to user

Object code : generated from above steps

Compilation linker : Integrate multiple source file to only one (generate an error if we have used same function name on 2 files ) **convert the .o to .ld**

Load file generated which is the file where multiple files are merged which is given input to loader

Compilation loader : Include all header / library files used in the program

Executable file : Target file

OS Loader : Allocate memory required for the process / application to run and

Convert the executable file into P**rocess**

Tokanizer ignores the keyword

Compiler can be of two types :

One pass compailer : Read the file from beg till end once

Two pass compailer : Read the file from beg till end twice

Address Binding two types

Compile time / Early : binding done before the linking

Run time address binding / Late binding : Binding done at time of linking

On first pass : every instruction will be written on the table on next pass it removes the instruction which are unnecessary and check if there are any errors and goes to second pass

On second pass : Based on the function we used in the program every thing else will be removed from the header expansion

gcc srcfile.c myapp.c -o myAppp

Name , Object , Size , - Symbol tablewhere all tokens are stored

Identifies the tokens written by used and write in Symbol table(token : function name)

Change to high level to assemnly code by code generator

If any header there expand that header – preprocessor

At last the compiler address binding done : For every instruction an address is generated and and the instruction is binded to it .

Binding the address to instruction done at last of Symbol table

Header is divided into 3 parts : Which is generated by **Compilation linker**

PE (Portable Executable ) Header :

Will have bit wise kind of information what bit compiler used , what bit OS it is , LSB or MSB these informations are stored

Static library dependency Symbol table:

Which hold the Dependencies of file Name , Object , Size , Usage

Start up routine : C runtime in Windows

CPU stars from this stage .

Here Environment is set for the application : App Environment Setup

App Environment Setup : This helps in assigning the values to global variables

This helps in storing the argument for the application **./a.out arg1 , arg2**

This at last calls the main function

This will call a function memset or memcpy (BSS , , ‘0’) will set the

uninitialized global and static and extern variable to 0

Code :

Place where all the functions are written with address in an instruction way

Preprocessor Puzzles :

Example 1 :

#include<stdio.h>

#define SQ(x) x\*x

int main()

{

int i=5 ;

int j=6 ;

printf("Hi\n");

printf("SQ = %d " , SQ(i+j)); // i + j \* i + j

}

O/P : 41

Explanation // i + j \* i + j

Example 2

#include<stdio.h>

void fucn()

{

#define TEN 10 // no error the preprocessor fetch the value at first

// error is when we are doing the definition again

}

void use\_function()

{

printf("TEN = %d " , TEN);

}

int main()

{

printf("Hi");

printf("\n");

use\_function();

}

O/P : 10

Explanation : // preprocessor at first define the macros

**Prg :**

A computer screen shot of a computer code

Description automatically generated

gcc -E sample.c

A screenshot of a computer

Description automatically generated

gcc -S filename.c // we find the assembly file for the c file

Example 3 :

#include<stdio.h>

#define TEST(x,y,z) x=100;y=200;z=300

int main()

{

printf("Hi");

printf("\n");

int i=10 , j=20 , k=30 ;

if(i<5)

TEST(i,j,k);

printf("%d %d %d " , i , j , k );

}

OP : 10 200 300

Explanation :Compiler says only if i<5 condition fails

i value remain unchanged and rem j and k are changes this is because of the blocks are missing

Example 4 :

#include<stdio.h>

#define A 10

#define B A // we are changing the name of the macro

// the value of the A is replaced at time of preprocessor

int main()

{

printf("B = %d " , B);

}

O/P: 10

Explanation : we are changing the name of the macro the value of the A is replaced at time of preprocessor

Macro :

#if 0

This is a comment ( logical comment )

#endif

Static lib : are introduced to load the extra features on application at time of execution of an application

Functions :

We can have multiple declaration but it is mandatory to have one definition

dlopen() is a function that helps us to load the function to memory when it is called

#include<dlfncntl.h>

dlopen(“c:\\some\_file ” , “function\_name”) // from that library we are using that function

We can call the main() function if we are code proper but the important is when main() function ends the program ends

Functions :

Function decaration / Prototype / Signature

Return name

Name

Argument

Scope

Exception throw

Calling conversions

Naming conversion / Name mangling / Function decoration

Function Definition

{ prologie | body | } Epilogue

**Process Parts :**

1. Code
2. Data
3. User Stack
4. Heap
5. Kernel Stack - stores the system calls is used in the process
6. PCB – Stores the status of the process where it begin and ends

**Program Counter / Instruction Pointer :** This helps in pointing to the code segment and helps in pointing to the next instruction to be fetched and executed

Data Segment:

Initialized -

Read only :

* + - When a viable is cont type the variable value cant be changed but in C and C+ + we can change it by pointer so the read only memory comes into picture
    - It can hold only the constants which are unchangeable

Read write :

* + - It can hold the global and extern variable which are initialized and modified

Static :

* + - Hold the static variable

Uninitialized -

Block Started by Symbols :

This holds the Global and External and Static variable which is not initialized which will be called by the memest function by start up routine

Extra Segment

When there is no space for the register where the data get stored it is stored on the Extra segment

When there is a function call and there is no register for the storage the compiler uses the Extra segment

**Stack and Heap :**

User stack grows from top to down :

EBP / ESP : External Stack Pointer / External Stack pointer always points to **User Stack top**

Heap memory grows from bottom to up :

IN user mode we cant read or write in Heap memory

We need to allocate a variable here so that we can read / write / free the memory of it

malloc , realloc , freealloc are used to work on this space

For changing the values stored on the Heap we can use pointer on the Stack segment or in the data segment

Pointer points to the free memory space are known as **Dangling Pointers** This happens when any data in the Heap segment is freed and try to access it through pointer

**Free store** help us to automatically delete the data when we handle the data

**Memory leak : When we have a data pointer by a pointer and the pointer is now pointing to somewhere and the pointer looses the address of the data . Now the data will be neither freed or used**

**Shared memory** is a place in between User Stack where it hold the memory of process1 which is needed by process2

**Calling convertion :**

stdcall : calle unwinds the stack for its arguments

cdecl : caller unwinds the stack for calley arguments (kgwars in python )

fastcall : Supported in VC++ no gcc

thiscall : from c++ onwards

**Naming Conversions :**

* C++ decorated the function name with the type of the argument passed and even with the return type of the function

In C: gfun or \_gfun

In C++ : gfun(type\_of\_arg)

In G++ : gfun X (return type) DH

* A function only with return type cant be overloaded because in the decoration it does not support the decorating with return type and the decoration done only based on name .

**Code the below program is for the C++ :**

#include<stdio.h>

/\*

.file "naming\_decoration.cpp"

.text

.globl \_Z4gfunv #Here

.def \_Z4gfunv; .scl 2; .type 32; .endef

.seh\_proc \_Z4gfunv

\_Z4gfunv:

when we have a function call with no argument it is decorated as gfunv (gfun)void argument

.globl \_Z4gfuni

.def \_Z4gfuni; .scl 2; .type 32; .endef

.seh\_proc \_Z4gfuni

\_Z4gfuni:

When we do a function call with argument in it . The name is decorated as (gfun)integer argument

this makes the difference bet C and C++ to make the C++ to do the function overloading

.seh\_endproc

.globl \_Z4gfund

.def \_Z4gfund; .scl 2; .type 32; .endef

.seh\_proc \_Z4gfund

\_Z4gfund:

.LFB47:

When we do a function call with arguments in it the name is decorated based on type of argument (gfun)double argument

\*/

void gfun()

{

int a=10;

}

void gfun(int a )

{

int b =a ;

}

void gfun(double a )

{

float b =a ;

}

int main()

{

gfun(); // function call

// when we have a function call with no argument it is decorated as gfunv (gfun)void argument

gfun(10);

// When we do a function call with argument in it . The name is decorated as (gfun)integer argument

gfun(11.12);

// When we do a function call with arguments in it the name is decorated based on type of argument (gfun)double argument

}

Types of Function call :

Call by value :

* When we pass the value as argument during a function call it is call by value

Call by reference

* When we pass the address as argument during a function call it is called call by reference

Inline function :

inline is similar to preprocessor way of execution and

it is a form of request to compiler it can also be ignored by compailer

Example :

inline int GETMAX(int x , int y ) // this is compailer function way

{

return(x>y?x:y);

}

**Inline :**

Advantage :

Function overloading with inline keyword gives all advantage of macro

Reduce the function call overhead

Disadvantage :

NOt generic if we give the double instead of int argument the floating point is ignored

- function overloading is the solution of this for GETMAXii to GETMAXdd

Example : for making the inline work similar to macro we are doing this

template<typename type1> //template is a blueprint

inline type1 GETMAX (type1 x , type1 y) // function overloading done here

{

return(x>y?x:y);

}

Compiler refuse the inline function on below occurrences

1. If a function contains a loop. (for, while and do-while)
2. If a function contains static variables.
3. If a function is recursive.
4. If a function return type is other than void, and the return statement doesn’t exist in a function body.
5. If a function contains a switch or goto statement.

Macros :

* Advantage :
  + inlined
  + Formula need to bec hanged at a palce
  + No need of funtion call frames of memory or context-switch
  + Generic - as it is generic it can be applied even for the doubl datatype
  + it imporoves performance
  + There is no need of prologue and epilogue
* Disadvantage :
  + Code segment blockage
  + Not that much intelligent

Example :

#define GETMAX(x,y) (x>y?x:y) // preprocessor way

Macro :

Day 3:

OOPS : Object Oriented Programming System.

Encapsulation

Abstraction

Message passing

Polymorphism

Static

Dynamic

Re-usabiity

Inheritance

Realization

Generalization

Association (uses)

Composition

Encapsulation :

Hiding the internal details and showing only the functionality it is encapsulation

In programming style definition { data + function } => class (entity)

* Data -> Data members , properties , attributes , value / weight , state
* Functions -> Member functions , methods (application based) , (UML) behaviour , (Mathematical ) Operation , (Realtime ) Behaviour

**Class :**

The data members and the member function of the class will be in the Heap memory because the class get memory during the run time so the memory will be dynamically allocated

We can have empty class in C++ because of the thiscall calling convertion it is not allowed in other languages because they require the size of the created entity

For a empty class the the size will be 1 bytes by default

The space is allocated when the class is instantiated, so 1 byte is allocated by the compiler to an object of an empty class for its unique address identification.

SOM : Simple object model

Constructor and destructor :

* These are necessary for the allocation and deallocation of the class memory when it is called by the another separate method main
* By default it will be in public side of a class for the main to get allocation and deallocation of that particular class
* Destructor not needed to be overloaded and it is not allowed
* Constructor can be overloaded

Copy constructor :

It usually makes the byte by byte copy

It degrades the performance

Assume always a copy constructor there

Special member functions :

Constructor

Destructor

Assignment operator

Userdefined datatype :

When user create any variable of his own type it is Userdefined datatype

Operator overloading :

We are overloading the operator

Assignments Day 3 :

Implement String operations :

1. Concatenation
2. Length
3. Compare
4. Copy
5. Convert
6. Reverse
7. Trim
8. Strip
9. Substring
10. Tokenization
11. Character find and replace
12. Search

Day 4 :

To make a member function as const apply const at the end

Const member function can call another const member function but not the non constant member function

Const variable :

The value of the const variable should not be changed

How to change the constant data member in a function

Const data member can be initialized in member function by initializer list

Initializer list can be only applied on constructor specialized member functions

Example :

A (int variable=-1 , int variable\_1=-1) : variable(variable) , variable\_1(variable\_1)

Where the left side is the global data member and right side is the local data member

Mutable variable :

Mutable is a keyword to make the variable to get modified the value even in const member function

Here the const\_cast is a function that helps in removing the variable constness in member function

Static member function is to be created where for any instance of the user definced data type call the

Heap

Free list is the list points to the element in the heap to free the particular space

Malloc , calloc , realloc in linux these are done by the brk system call

Interview :

malloc with 0 as size : malloc(sizeof(0)) // it will consume 4 bytes 0 is considered as the unsigned int

How to check the pointer points to valid address ?

We have to check it points to the NULL address then it is a valid address

Day 5 :

New keyword

Pointer will be in stack segment and the data of the pointer will be in the data readonly segment

Debugger line by line debug :

G - GNU (debugger) command line kind of debugger

This wont have the source code in the executable file

Much less in size

D -

B

The way to opern GDB

PS C:\C++ Training\Day 5> g++ -g .\overloaded\_new\_operator.cpp

PS C:\C++ Training\Day 5> gdb .\a.exe

GNU gdb (GDB) 11.2

/\*Copyright (C) 2022 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Type "show copying" and "show warranty" for details.

This GDB was configured as "x86\_64-w64-mingw32".

Type "show configuration" for configuration details.

For bug reporting instructions, please see:

<https://www.gnu.org/software/gdb/bugs/>.

Find the GDB manual and other documentation resources online at:

<http://www.gnu.org/software/gdb/documentation/>. \*/

For help, type "help".

Type "apropos word" to search for commands related to "word"...

Reading symbols from .\a.exe...

(gdb) list List ( or ) l : will give 10 lines

(gdb) break main # this will tell the line number of the main

**break or b can be done :**

Ex :

(gdb) break display()

Breakpoint 3 at 0x14000291c: file .\overloaded\_new\_operator.cpp, line 21.

(gdb) break 65 # we can also break line number

Breakpoint 4 at 0x140001593: file .\overloaded\_new\_operator.cpp, line 65.

(gdb) run # r (or) run helps to us to start the execution

Ex :

(gdb) run

Starting program: C:\C++ Training\Day 5\a.exe

[New Thread 25608.0x55e0]

[New Thread 25608.0x4214]

[New Thread 25608.0x630c]

Thread 1 hit Breakpoint 1, main () at .\overloaded\_new\_operator.cpp:88

88 group1::main();

(gdb) step # step (or) s will step into the function

28 cout <<"ptr holds " << ptr <<"\n";

Ex :

group1::main () at .\overloaded\_new\_operator.cpp:43

43 cout <<"ptr after new keyword holds " << ptr <<"\n";

(gdb) step

ptr after new keyword holds 0x15130a44d28

45 if(NULL == ptr )

(gdb) step

54 ptr->display();

(gdb) step

Thread 1 hit Breakpoint 3, group1::A::display (this=0x15130a44d28) at .\overloaded\_new\_operator.cpp:21

21 cout<<"From " << this <<" The value of variable in class is :" << variable << " \n";

(gdb) step

From 0x15130a44d28 The value of variable in class is :-1

22 }

(gdb) step

group1::main () at .\overloaded\_new\_operator.cpp:56

56 ptr[1].display();

(gdb) step

Thread 1 hit Breakpoint 3, group1::A::display (this=0x15130a44d2c) at .\overloaded\_new\_operator.cpp:21

21 cout<<"From " << this <<" The value of variable in class is :" << variable << " \n";

(gdb) step

From 0x15130a44d2c The value of variable in class is :-1

22 }

(gdb) print ptr # p <var> or print <var > for printing var

(gdb) backtrace # backtrace (or) bt

(gdb) next #

Hello World :

PS C:\C++ Training\Day 5> g++ -g .\hello\_world\_GDB.cpp

PS C:\C++ Training\Day 5> gdb .\a.exe

GNU gdb (GDB) 11.2

Copyright (C) 2022 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Type "show copying" and "show warranty" for details.

This GDB was configured as "x86\_64-w64-mingw32".

Type "show configuration" for configuration details.

For bug reporting instructions, please see:

<https://www.gnu.org/software/gdb/bugs/>.

Find the GDB manual and other documentation resources online at:

<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".

Type "apropos word" to search for commands related to "word"...

Reading symbols from .\a.exe...

(gdb) run

Starting program: C:\C++ Training\Day 5\a.exe

[New Thread 6600.0x6334]

[New Thread 6600.0x55b4]

[New Thread 6600.0x920]

Hello World

[Thread 6600.0x6334 exited with code 0]

[Thread 6600.0x920 exited with code 0]

[Thread 6600.0x55b4 exited with code 0]

[Inferior 1 (process 6600) exited normally]

Day 6 :

Inheritance types :

Single : class1<-class2

Multiple : class 1-<class 2 class1<-class3

Multi level : class1 <- class2 -<class3

Hierarchical

Hybrid or Diamond inheritance : multilevel + multiple

Virtual inheritance : It is inheritance at runtime

Parameter : during the function definition

Argument : during the function call

Access modifier

Private

Class derived : private base

|  |  |
| --- | --- |
| Base | Derived |
| Public | Private |
| Private | Private |
| Protected | Private |

Protected

Class derived : protected base

|  |  |
| --- | --- |
| Base | Derived |
| Public | Protected |
| Private | Private |
| Protected | Protected |

Public

Class derived : public base

|  |  |
| --- | --- |
| Base | Derived |
| Public | Public |
| Private | private |
| Protected | protected |

Private :

Only base class member functions can access the private data members of the class

Protected :

Derived can access the protected member functions or data members and the specifier will be based on how we extends

Public :

Access by all of the member functions in the derived class

Class :

If the base clss is declared using class keyword **class**

If no access specifier inside the class all are considered as private

If we didt specify the access specifier while inheriting a class by default it is considered as **private**

Struct :

If the base clss is declared using struct keyword **struct**

If no access specifier inside the struct all are considered as public

If we didt specify the access specifier while inheriting a struct now the access specifier is by default public

Constructor call at the initializer list will consider as the constructor class or else it wont take as constructor call rather it takes it as a temporary object

Day 7

The size of the empty class is 1 byte and when the other call is inherited from the base class the size of the derived class will be same 1 byte where it hold the base address of it

Polymorphism

Function overriding : when a function is called from which is in the base class and the derived is not having a function same like that . Then compiler search for the function in the derived first and if it is not there then it finds form the base class

Order of preference :

Derived class first if the function is not there then search in base class

SOLID :

S -> SRP Single Responsibility principle

A class must have function not more than 1 and not less than 1

O -> open closed principle

Open for inheritance Closed for the modification.

L - > Liskow Substitution Principle

Base class pointer can to derived object but not he derived class pointer cant point to the base class object

Error :

PS C:\C++ Training\Day 7> g++ .\liskov\_substitution\_principle.cpp

.\liskov\_substitution\_principle.cpp: In function 'int main()':

.\liskov\_substitution\_principle.cpp:42:18: error: invalid conversion from 'A\*' to 'B\*' [-fpermissive]

42 | derived\_ptr= &base\_object;

| ^~~~~~~~~~~~

| |

| A\*

A \* base\_ptr = NULL ;

base\_ptr = &derived\_object;

// though it get represent to the derived class object it can only access the base class data members and member functions

// base\_ptr->derived\_function();

base\_ptr->base\_function();

Static polymorphism / Function Overloading

At compile time it chooses which function to be chosen based on the type

Dynamic polymorphism / Function Overloading

Ignoring the base class function override on the derived class fucntion

Day8 :

We cant virtual the constructor as it is mandatorily get called when a object is created

When u have a virtual member function and u don’t have the special member function compiler will automatically call the base destructor

Golden rules :

Base class not know anything about derived

Special member functions will initializa the vptr

Golden rules are applied on the Special member function only

Special member functions are :

Constructor

Destructor

Operator =

Virtual function and Vtable understanding

Vtable is a virtual table that will be created when it finds virtual inside a class and it holds the virtual function inside the class in that vtable for a class table

In that Vtable the entry of the class will be of the order of declaration

If the same function in the base is same in derived then the derived function is also considered as the virtual function and a Vtable for the derived class will also be created where it hold the virtual function in it

New understanding

class A

{

    int admno ;

    char \*sname;

    float eng , math, science ;

    float total ;

    float ctotal()

    {

        return (eng+math+science);

    }

    public :

        void takedata(int a ,char \*array , float e , float sci)

        {

            int count;

            int size = strlen(array) + 1 ;

            sname = new char[size];

            for(int i=0;array[i]!='\0';i++)

            {

                sname[i]=array[i];

            }

            admno = a;

            eng = e;

            science = sci ;

        }

        int result = ctotal();

        void showdata()

        {

            cout <<"Addno = " << admno << " sname = " <<sname  <<" eng =" << eng << " science = " << science <<"\n";

        }

};

int main()

{

    A object ;

    // char array="Akshay";  //working proper

    // char array[]="Akshay"; not working proper why

    object.takedata(10,"Akshay", 50.34 , 23.32);

    object.showdata();

}

In this program we are dynamically allocating the memory for that array elements instead of wasting the memory

When the virtual function is called in the constructor the vtable will be initialized with that value or else it will be initialized partially and linker will take care of initialization

Destructor can be even virtualized and the virtualization will follow the derived class destructor because having the same signation and prototyping

Shallow copy :

In shallow copy, an object is created by simply copying the data of all variables of the original object. This works well if none of the variables of the object are defined in the heap section of memory. If some variables are dynamically allocated memory from heap section, then the copied object variable will also reference the same memory location.

This will create ambiguity and run-time errors, dangling pointer. Since both objects will reference to the same memory location, then change made by one will reflect those change in another object as well.

A screenshot of a computer

Description automatically generated

|  |
| --- |
| Program to understand the Shallow Copy  #include <iostream>  #include<cstring>  using namespace std;  //headerfile.h  class Base{  private:  char \* private\_dm\_ptr;  public:  Base();  Base(const char \*ptr);  Base(Base & rhs);  ~Base();  };  // definations or headerfile  Base::Base() { cout<<"Base:: empty string has been initiated. this ="<<this<<"\n";}  Base::Base(const char \*ptr){  cout<<"Base(const char \*ptr:string )object initiated. this="<<this<<"\n";  private\_dm\_ptr = new char(strlen(ptr) +1);  strcpy(private\_dm\_ptr, ptr); // copy the string to that address  }  Base::Base(Base & ref )  {  this->private\_dm\_ptr = ref.private\_dm\_ptr ;  }  Base::~Base() {  cout<<"~Base:: empty string has been initiated. this ="<<this<<"\n";  if(private\_dm\_ptr){  cout<<"initiated Base is setting to free...private\_dm\_ptr"<<private\_dm\_ptr<<"\n";  delete [] private\_dm\_ptr;  cout<<" +- +- +- +- +- +- +- +- +- \n";  }  }  // application-side  int main()  {  cout<<"-------------\n";  Base str1; //default construcor  cout<<"-------------\n";  Base str2("hello"); // single-parametarized construcor    cout<<"-------------\n";  Base str3(str2); //copy-construcotr  }  /\*  this copy constructor  The data member is accessible for all objects with same value  \*/ |

Deep copy :

Deep copy, an object is created by copying data of all variables, and it also allocates similar memory resources with the same value to the object. In order to perform Deep copy, we need to explicitly define the copy constructor and assign dynamic memory as well, if required. Also, it is required to dynamically allocate memory to the variables in the other constructors, as well.

A screenshot of a computer

Description automatically generated

|  |
| --- |
| Program to understand the Deep Copy  #include <iostream>  #include<cstring>  using namespace std;  //headerfile.h  class Base{  private:  char \* private\_dm\_ptr;  public:  Base();  Base(const char \*ptr);  Base(Base & rhs);  ~Base();  void display ()  {  cout <<"MyString is : " << private\_dm\_ptr <<"\n";  }  };  // definations or headerfile  Base::Base() { cout<<" @ Constructor this ="<<this<<"\n";} //Base:: empty string has been initiated.  Base::Base(const char \*ptr){  cout<<" @ Single Parameterized Constructor this="<<this<<"\n"; //Base(const char \*ptr:string )object initiated.  private\_dm\_ptr = new char(strlen(ptr) +1);  strcpy(private\_dm\_ptr, ptr); // copy the string to that address  }  Base::Base(Base & ref )  {  cout<<" @ Copy Constructor this="<<this<<"\n";  if(ref.private\_dm\_ptr)  {  this->private\_dm\_ptr = new char[strlen(ref.private\_dm\_ptr) +1];  strcpy(this->private\_dm\_ptr,ref.private\_dm\_ptr);  }  // move constructor  // below we are transferring the ownership  // this->private\_dm\_ptr = ref.private\_dm\_ptr;  // ref.private\_dm\_ptr = NULL ;  else{  this->private\_dm\_ptr = ref.private\_dm\_ptr ; // here ref.private\_dm\_ptr which is holding the NULL  }  }  Base::~Base() {  cout<<" @ Destructor this ="<<this<<"\n"; //~Base:: empty string has been initiated.  if(private\_dm\_ptr){  cout<<"Freeing private\_dm\_ptr going on "<<private\_dm\_ptr<<"\n";  delete [] private\_dm\_ptr;  cout<<" +- +- +- +- +- \n";  }  }  // application-side  int main()  {  cout<<"-------------\n";  // Base str1; //default construcor  cout<<"-------------\n";  Base str2("hello"); // single-parametarized construcor    cout<<"-------------\n";  Base str3(str2); //copy-construcotr  str2.display();  str3.display();  }  /\*  this deep copy constuctor will make the seperate copy of the datamember  this move constructor will transfer the ownership from the str2 to str3  \*/ |

Operator overloading :

Operator can be overloaded in C++ with a specific function for it in the program

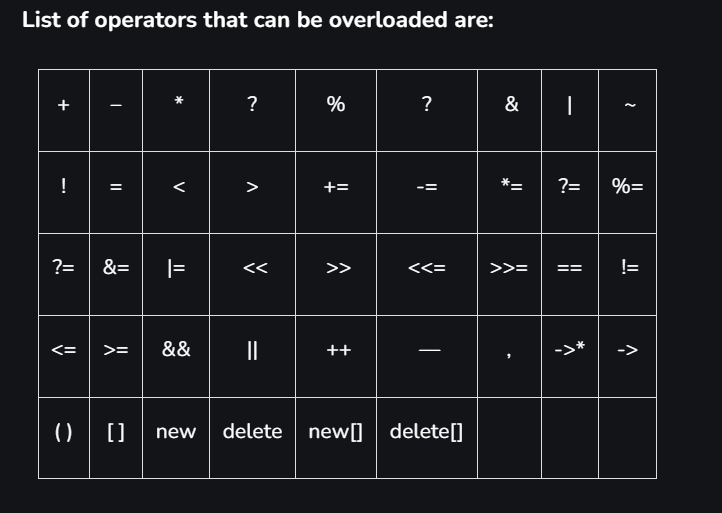
Here the name of the function operator cant be changed

|  |
| --- |
| Program to understand Operator overloading  #include<iostream>  using namespace std ;  class A  {    friend void operator + (int args , A & ref );  friend void operator + (A & ref ,int args ) ;    public:  int private\_integer=0;  // int private\_integer=0;  // Commenting the below to show operator overloading in diff way globally  // void operator + (int args )  // {  // private\_integer = private\_integer + args ;  // cout<<"The data member value is : " << private\_integer <<"\n" ;  // }  void display\_values()  {  cout<<"The data member value is : " << private\_integer <<"\n" ;  }  void operator = (A & ref )  {  this->private\_integer = ref.private\_integer;  }  //We are going to do ++obj mean increment the datamember obj++ mean decremernt the datamember  //operator overloading  void operator ++()  {  private\_integer = private\_integer +1 ;  cout<<"After the increment : " << private\_integer <<"\n";  }  //for diff we are creating the argument for overloading to be done  void operator ++(int args) // for diffrentiation we are putting the arguments  {  private\_integer = private\_integer - 1 ;  cout<<"After the increment : " << private\_integer <<"\n";  }  };  //(2 + object ) another global fucntion that act as the friend function in class and do a operator overloading  void operator + (int args , A & ref )  {  ref.private\_integer= + args ;  cout<<"The data member value is : " << ref.private\_integer <<"\n" ;  }  // (object + 6) another global fucntion that act as the friend function in class and do a operator overloading  void operator + (A & ref ,int args )  {  ref.private\_integer= + args ;  cout<<"The data member value is : " << ref.private\_integer <<"\n" ;  }  int main()  {  A object ;  A object\_1 ;    //Here we are doing the + operation overloading  5+object ;  6+object\_1;  A object\_2;  //Here we are doing the = operation overloading  object\_2 = object\_1; // here the assignment operation done and the CPU cycles wil be 4 because of the one data member (prev ) (when the = operator applied)  //now we are creating the assignment operator overloading fucntion where we are copying the values    object\_2.display\_values();  object\_1.display\_values();  //here we are overloading the pre increment and post increment operator  ++object\_2;  object\_2++; // this will take the default argument as (0)    } |

Here in this program we are overloading the operator by creating a function for it . Here instead of going with the operation work the compiler will do a function call to overload that operator .

Day 9

Operator overloading :



List of operators that cannot be overloaded

1) Scope Resolution Operator (::)

2) Ternary or Conditional Operator (?:)

3) Member Access or Dot operator (.)

4) Pointer-to-member Operator (.\*)

5) Object size Operator (sizeof)

6) Object type Operator(typeid)

7) static\_cast (casting operator)

8) const\_cast (casting operator)

9) reinterpret\_cast (casting operator)

10) dynamic\_cast (casting operator)

1.completed name =

2. Access Specifiers

3. Inner/Nested class is friend by default

4. Outer Template-Typer can be used in Nested/Inner class

Static

Dynamic

Reinterpret

Const

C style typecasting is very dangerous as we can typecast to any type without the source knowledge

Const cast:

For converting the const pointer to normal pointer (removing the const )

Dynamic Cast :

when there is a downcasting from derived to base then static\_cast done

The program must be polymorphic so we have to put the virtual to make the program as polymorphic and to make all class as virtual we apply virtual at destructor

Static cast :

when there is a upcasting from derived to base then static\_cast done

Day 10

Copy Constructor

Member by member copy instead of byte by byte copy

Const\_Cast <destination> (variable ):

If it can convert the variable to destination it will or else it will give NULL

When a class is virtual a vtable and class table will be created and class loader hold the class declarations

Copy constructor can be called by the () and the = assignment operator

I=j

J data is copied to I location

Cout<< operator ;

The overload has to be in the global function because the cout is of the customized datatype

Default constructor :

A obj() // this will thought as a function by the compiler

Reason :

In global we are having the function takes void and return a object ;

A function(void ); // it is a global function

class Seminar

{

int time;

public:

Seminar() //Function 1

{

time = 30;

cout << "Seminar starts now" << endl;

}

void lecture() //Function 2

{

cout << "Lectures in the seminar on" << endl;

}

Seminar(int duration) //Function 3

{

time = duration;

cout << "Seminar starts now" << endl;

}

~Seminar() //Function 4

{

cout << "Thanks" << endl;

}

};

i. Write statements in C++ that would execute Function 1 and Function 3 of class

Seminar.

Seminar object ();

Seminar object (int a );

ii. In Object Oriented Programming, what is Function 4 referred as and when does it

get invoked/called?

Destructor

At time of ending the object scope

iii. In Object Oriented Programming, which concept is illustrated by Function 1 and

Function 3 together? default and single parameterized constructor

2

class Test

{

char paper[20];

int marks;

public:

Test () // Function 1

{

strcpy (paper, "Computer");

marks = 0;

}

Test (char p[]) // Function 2

{

strcpy(paper, p);

marks = 0;

}

Test (int m) // Function 3

{

strcpy(paper,"Computer");

marks = m;

}

Test (char p[], int m) // Function 4

{

strcpy (paper, p);

marks = m;

}

};

i. Write statements in C++ that would execute Function 1, Function 2, Function 3 and

Function 4 of class Test.

Test object();

Test object(“Akshay”);

Test object(9);

Test object(“Baalaji”,9);

ii. Which feature of Object Oriented Programming is demonstrated using Function 1,

Function 2, Function 3 and Function 4 together in the above class Test? Function overloading

3.

class Sample

{

private:

int x;

double y;

public :

Sample(); //Constructor 1

Sample(int); //Constructor 2

Sample(int, int); //Constructor 3

Sample(int, double); //Constructor 4

};

i. Write the definition of the constructor 1 so that the private member variables are

initialized to 0.

Sample() : x(0) , y(0)

ii. Write the definition of the constructor 2 so that the private member variable x is

initialized according to the value of the parameter, and the private member variable

y is initialized to 0.

Sample(int args=-1) : x(args) , y(0)

iii. Write the definition of the constructors 3 and 4 so that the private

member variables are initialized according to the values of the parameters.

Sample(int args =-1 , int args1 = -2 ) : x(args) , y(args1)

Sample(int args =-1 , double args1 = -2 ) : x(args) , y(args1)

Day 11 :

Operator Overlading :

Subscript operator overloading :

Single dimensional array formula :

Base address + (index \* sizeof(datatype) )

Multi dimensional array formula :

Base address + (row\_index \* sizeof(datatype) ) + (column\_index \* sizeof(datatype) )

Ex :

arr[3][4]

arr[3] =>Row index = 1000+ 4\*(3\*4) = 1048

arr[2]=> Row Index = 1000 + 4\*(2\*4) = 1032 {here 4 is length of the array row size}

arr[2][1] = (1000) + (4\*(2\*4)) + (1\*4) = 1000 + 32 + 4 = 1036

1000 + 4(2\*4)

Only the function with return type cant be overloaded

4 inner class charecterstics :

Complete name

Access specifier

Inner / nested class is friend by default to parent class

Outer template – type can be used in nested and inner class

Containement :

Containement is classA is there and a classB is there and these are not inherited and we are thinking to take the features from the classA from the classB this can be done by creating a object of classA from the classB

Day 12 :

New operator can be overloaded

:: cant be overloaded

Presedence cant be changed by overloading

A white rectangular box with green and red text

Description automatically generated

Associations :

Containment :

This is a concept where the classB which is individual class and the classA is a individual class now the classB using the classA features (member function ) by creating the object of classA .

This is known as has a relationship

Composition :

Containment : We know the state at first (When one is born other is born )

Aggregation : Don’t know the state at first (When one born and other is not related to this )

Tightly :

Loosely : (When created we use the obj and set it free to be used by some other object )

Linked list :

Double is given the most priority thamn the float