* ***Purpose :*** **Classwork.**

* ***Date :*** **30/12/2025**

* ***Author :*** **Vikas Srivastava**

* ***ID :*** **55984**

* ***Batch ID :*** **25SUB4505**

---

1. ***Code :*** Write a program to demonstrate the order of execution of constructors and destructors in multiple inheritance in C++, showing that base class constructors are called first (in inheritance order), followed by the derived class constructor, and destructors are called in the reverse order when the object goes out of scope.

inherCtorDemo.cpp

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

struct A{
  A(){cout<<"A()"<<endl; }
  ~A(){cout<<"~A()"<<endl; }
};

struct B{
  B(){cout<<"B()"<<endl; }
  ~B(){cout<<"~B()"<<endl; }
};

struct C{
  C(){cout<<"C()"<<endl; }
  ~C(){cout<<"~C()"<<endl; }
};

struct D: A, B, C{
  D():B(), C(), A(){cout<<"D()"<<endl; }
  ~D(){cout<<"~D()"<<endl; }
};

int main(){
  D dObj;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\inherCtorDemo.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
A()
B()
C()
D()
~D()
~C()
~B()
~A()
```

2. ***Code :*** Write a program to demonstrate the order of constructor and destructor execution in single inheritance in C++, where the base class constructor executes first, followed by the derived class constructor, and destructors are called in reverse order when the object goes out of scope.

inherCtorOne.cpp

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

class Base{
public:
  Base() {cout<<"Base::Base()"<<endl; }
  ~Base() {cout<<"Base::~Base()"<<endl; }
};

class Derived: public Base{
public:
  Derived() {cout<<"Derived::Derived()"<<endl; }
  ~Derived() {cout<<"Derived::~Derived()"<<endl; }
};

int main(){
  Derived dObj;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\inherCtorOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
Base::Base()
Derived::Derived()
Derived::~Derived()
Base::~Base()
```

3. ***Code :*** Write a program to demonstrate constructor chaining in C++ inheritance, where a derived class constructor explicitly calls a parameterized base class constructor using an initializer list, and observe the order of constructor and destructor execution.

inherCtorTwo.cpp

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

class Base{
public:
  Base() {cout<<"Base::Base()"<<endl; }
  Base(int) {cout<<"Base::Base(int)"<<endl; }
  ~Base() {cout<<"Base::~Base()"<<endl; }
};

class Derived: public Base{
public:
  Derived() {cout<<"Derived::Derived()"<<endl; }
  Derived(int x):Base(x) {cout<<"Derived::Derived("<<x<<")"<<endl; }
  ~Derived() {cout<<"Derived::~Derived()"<<endl; }
};

int main(){
  Derived dObj=100;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\inherCtorTwo.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
Base::Base(int)
Derived::Derived(100)
Derived::~Derived()
Base::~Base()
```

4. ***Code :*** Write a program to demonstrate multilevel inheritance in C++, where a class is derived from another derived class, allowing the grandchild class to access the member functions of both its parent and grandparent classes, showcasing hierarchical access in object-oriented programming.

multilevelInherOne.cpp

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

class Base{ //Parent
public:
  void funOne(){cout <<"Base::funOne()"<<endl;}
  void funTwo(){cout <<"Base::funTwo()"<<endl;}
};

class Derived: public Base{ //Child
public:
  void funThree() {cout <<"Derived::funThree"<<endl; }
  void funFour() {cout <<"Derived::funFour"<<endl; }
};

class DerivedOne: public Derived{//Grand-child
public:
  void funFive() {cout<<"DerivedOne::funFive"<<endl; }
  void funSix() {cout<<"DerivedOne::funSix"<<endl; }
};

int main(){
  DerivedOne dObj;
  dObj.funOne();
  dObj.funTwo();
  dObj.funThree();
  dObj.funFour();
  dObj.funFive();
  dObj.funSix();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\multilevelInherOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
Base::funOne()
Base::funTwo()
Derived::funThree
Derived::funFour
DerivedOne::funFive
DerivedOne::funSix
```

5. ***Code :*** Write a program to demonstrate multiple inheritance in C++, where a derived class inherits from two different base classes, allowing the derived object to access member functions of both base classes along with its own functions.

multipleInherOne.cpp

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

class BaseOne{
public:
  void funOne(){cout <<"BaseOne::funOne()"<<endl;}
  void funTwo(){cout <<"BaseOne::funTwo()"<<endl;}
};

class BaseTwo{
public:
  void funThree(){cout <<"BaseTwo::funThree()"<<endl;}
  void funFour(){cout <<"BaseTwo::funFour()"<<endl;}
};

class Derived: public BaseOne, public BaseTwo{ //having 2 base classes
public:
  void funFive() {cout <<"Derived::funFive()"<<endl; }
  void funSix() {cout <<"Derived::funSix()"<<endl; }
};

int main(){
  Derived dObj;
  dObj.funOne();
  dObj.funTwo();
  dObj.funThree();
  dObj.funFour();
  dObj.funFive();
  dObj.funSix();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ multipleInherOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
BaseOne::funOne()
BaseOne::funTwo()
BaseTwo::funThree()
BaseTwo::funFour()
Derived::funFive()
Derived::funSix()
```

6. ***Code :*** Write a program to demonstrate how function name ambiguity in multiple inheritance can be resolved using the using declaration, allowing specific base class functions to be directly accessible in the derived class.

multipleInherTwo.cpp

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

class BaseOne{
public:
  void funOne(){cout <<"BaseOne::funOne()"<<endl;}
  void funTwo(){cout <<"BaseOne::funTwo()"<<endl;}
};

class BaseTwo{
public:
  void funOne(){cout <<"BaseTwo::funOne()"<<endl;}
  void funTwo(){cout <<"BaseTwo::funTwo()"<<endl;}
};

class Derived: public BaseOne, public BaseTwo{ //having 2 base classes
public:
  using BaseOne::funOne;
    using BaseTwo::funTwo;
  void funThree() {cout <<"Derived::funThree()"<<endl; }
  void funFour() {cout <<"Derived::funFour()"<<endl; }
};

int main(){
  Derived dObj;
  dObj.funOne();
  dObj.funTwo();
  dObj.funThree();
  dObj.funFour();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\multipleInherTwo.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
BaseOne::funOne()
BaseTwo::funTwo()
Derived::funThree()
Derived::funFour()
```

7. ***Code :*** Write a program to demonstrate operator overloading in C++, where arithmetic operators (+ and -) are overloaded using friend functions to perform operations on objects of a user-defined class, along with overloading the stream insertion (<<) operator to display object data.

operatorFour.cpp

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

class Num{
  int data;
public:
  Num(int x=0): data(x) {}
  friend Num operator+(const Num&, const Num&);
  friend Num operator-(const Num&, const Num&);
  friend ostream& operator<<(ostream &, const Num &);
};

Num operator+(const Num& lhs, const Num& rhs){
  Num temp(lhs.data + rhs.data);
  return temp;
}

Num operator-(const Num& lhs, const Num& rhs){
  Num temp(lhs.data - rhs.data);
  return temp;
}

ostream& operator<<(ostream &out, const Num &obj){
  out<<"data: "<<obj.data;
  return out;
}

int main(){
  Num a = 100, b = 20;
  cout<<a<<"   "<<b<<endl;
  Num c = a + b;
  Num d = c + a - b;
  cout<<c<<"   "<<d<<endl;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\operatorFour.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
data: 100   data: 20
data: 120   data: 200
```

8. Code : Write a program to demonstrate overloading of global new and delete operators in C++, showing how memory allocation and deallocation can be customized globally, and how object construction and destruction occur alongside these custom memory management functions.

operatorNewOne.cpp

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

void * operator new(size_t size){
  cout<<"Global operator new with size: "<<size<<endl;
  void *ptr = malloc(size);
  if (ptr == NULL){
    throw bad_alloc();
  }
  return ptr;
}

void operator delete(void *ptr){
  cout<<"Global operator delete"<<endl;
  free(ptr);
}

struct Test{
  int data;
  Test(int x=0):data(x){cout<<"Test()"<<endl;}
  ~Test(){cout<<"~Test()"<<endl;}
};

int main(){
  Test *ptr = new Test();
  delete ptr;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\operatorNewOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
Global operator new with size: 4
Test()
~Test()
```

9. ***Code :*** Write a program to demonstrate overloading new and delete operators at the class level in C++, allowing a class to control its own memory allocation and deallocation, separate from the global new and delete operators.

operatorNewTwo.cpp

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

class Test{
  int data;
public:
  Test(int x=0): data(x) {}

  void *operator new(size_t size){
    cout<<"Test::operator new size: "<<size<<endl;
    void *ptr = malloc(size);
    if (ptr == NULL)
      throw bad_alloc();
    return ptr;
  }

  void operator delete(void *ptr){
    cout<<"Test::operator delete "<<endl;
    free(ptr);
  }
};

int main(){
  Test *ptr = new Test(100);
  delete ptr;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ operatorNewTwo.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
Test::operator new size: 4
Test::operator delete
```

10. ***Code :*** Write a program to demonstrate operator overloading for pre-increment (++a) and post-increment (a++) in C++, showing the difference in behaviour and return values when these operators are applied to objects of a user-defined class.

operatorOne.cpp

```cpp
#include <iostream>
using namespace std;
class Num{
  int data;
public:
  Num(int x=0): data(x) {}
  Num& operator++(){
    cout<<"Pre-fix"<<endl;
    data++;
    return *this;
  }
  Num operator++(int){
    cout<<"Post-fix"<<endl;
    Num temp(*this);//store previous data here
    data++;
    return temp; //return non-modified data
  }
  void disp(ostream &out){
    out<<"data: "<<data<<endl;
  }
};
int main(){
  Num a = 10;
  ++a;
  a.disp(cout);
  a++;
  a.disp(cout);
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\operatorOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
Pre-fix
data: 11
Post-fix
data: 12
```

11. Code : Write a program to demonstrate operator overloading using member functions in C++, where arithmetic operators (+ and -) are overloaded inside a class to perform operations on objects, and the stream insertion operator (<<) is overloaded to display object data.

operatorThree.cpp

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

class Num{
  int data;
public:
  Num(int x=0): data(x) {}
  Num operator+(const Num& rhs){
    Num temp(data + rhs.data);
    return temp;
  }

  Num operator-(const Num& rhs){
    Num temp(data - rhs.data);
    return temp;
  }
  friend ostream& operator<<(ostream &, const Num &);
};

ostream& operator<<(ostream &out, const Num &obj){
  out<<"data: "<<obj.data;
  return out;
}

int main(){
  Num a = 100, b = 20;
  cout<<a<<"   "<<b<<endl;
  Num c = a + b;
  Num d = c + a - b;
  cout<<c<<"   "<<d<<endl;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\operatorThree.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
data: 100   data: 20
data: 120   data: 200
```

12. Code : Write a program to demonstrate operator overloading using member functions in C++, where arithmetic operators (+ and -) are overloaded to operate on objects of a class, and a separate display function is used to print object data instead of overloading the stream insertion operator.

operatorTwo.cpp

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

class Num{
  int data;
public:
  Num(int x=0): data(x) {}
  Num operator+(const Num& rhs){
    Num temp(data + rhs.data);
    return temp;
  }

  Num operator-(const Num& rhs){
    Num temp(data - rhs.data);
    return temp;
  }
  void disp(ostream &out){
    out<<"data: "<<data<<endl;
  }
};

int main(){
  Num a = 100, b = 20;
  a.disp(cout);
  b.disp(cout);
  Num c = a + b;
  Num d = c + a - b;
  c.disp(cout);
  d.disp(cout);
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\operatorTwo.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
data: 100
data: 20
data: 120
data: 200
```

13. Code : Write a program to demonstrate runtime polymorphism in C++ using a virtual function, where a base class pointer calls the appropriate overridden function based on whether it points to a base class object or a derived class object at runtime.

polyRuntimeOne.cpp

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

class Base{
public:
  virtual void funOne(){cout <<"Base::funOne()"<<endl; }
};

class Derived: public Base{
public:
  void funOne() {cout<<"Derived::funOne()"<<endl;}
};

int main(){
  //Remember Derived is a Base class
  Base *ptr, bObj;
  Derived dObj;

  ptr = &bObj; //Base class object's address is assigned to Base class Pointer
  ptr->funOne();

  ptr = &dObj;//Derived class Object's address is assigned to Derived Class Pointer
  ptr->funOne();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ polyRuntimeOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
Base::funOne()
Derived::funOne()
```

14. Code : Write a program to demonstrate access specifiers in C++ (private, protected, and public) and how they behave during inheritance, showing that private members are not accessible, protected members are accessible within derived classes, and public members are accessible everywhere through derived class objects.

protectedMember.cpp

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

class Test{
//private member by default
  void funOne(){cout <<"Test::funOne()--> Private"<<endl; }
protected:
  void funTwo(){cout <<"Test::funTwo() --> Protected"<<endl; }
public:
  void funThree(){cout <<"Test::funThree() --> Public"<<endl; }
};

class Derived: public Test{
public:
  void funFour(){
    //funOne(); //private member
    funTwo(); //protected member inherited here
    funThree();//public member inherited here
  }
};

int main(){
  Derived dObj;
  dObj.funThree();
  dObj.funFour();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\protectedMember.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
Test::funThree() --> Public
Test::funTwo() --> Protected
Test::funThree() --> Public
```

15. *Code :* Write a program to demonstrate single inheritance in C++, where a derived class inherits public member functions from a base class and can access both the inherited functions and its own member functions using a derived class object.

simpleInherOne.cpp

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

class Base{
public:
  void funOne(){cout <<"Base::funOne()"<<endl;}
  void funTwo(){cout <<"Base::funTwo()"<<endl;}
};

class Derived: public Base{
public:
  void funThree() {cout <<"Derived::funThree"<<endl; }
  void funFour() {cout <<"Derived::funFour"<<endl; }
};

int main(){
  Derived dObj;
  dObj.funOne();
  dObj.funTwo();
  dObj.funThree();
  dObj.funFour();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\simpleInherOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
Base::funOne()
Base::funTwo()
Derived::funThree
Derived::funFour
```

16. *Code :* Write a program to demonstrate string tokenization in C using the strtok() function, where a sentence is split into individual words based on a delimiter (space) and each token is printed separately.

staticCStyle.c

```c
#include <stdio.h>
#include <string.h>


int main(){
  char str[]="One way of using a class is by creating objects and using the member through that object";
  char *ptr=strtok(str, " ");//FIRST time giving address
  while(ptr!=NULL){
    printf("%s\n", ptr);
    ptr = strtok(NULL, " ");//consecutive no address given
  }
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> gcc .\staticCStyle.c
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
One
way
of
using
a
class
is
by
creating
objects
and
using
the
member
through
that
object
```

17. ***Code :*** Write a program to demonstrate string tokenization in C++ using the strtok() function from the C string library, where a sentence is split into words based on a space delimiter and each token is printed on a new line.

staticCStyle.cpp

C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork\staticCStyle.cpp (preview)

```cpp
#include <cstring>
using namespace std;

int main(){
  char str[]="One way of using a class is by creating objects and using the member through that object";
  char *ptr=strtok(str, " ");//FIRST time giving address
  while(ptr!=NULL){
    cout<<ptr<<endl;
    ptr = strtok(NULL, " ");//consecutive no address given
  }
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\staticCStyle.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
One
way
of
using
a
class
is
by
creating
objects
and
using
the
member
through
that
object
```

18. Code : Write a program to demonstrate the use of a static local variable inside a function, where the variable retains its value between multiple function calls and continues incrementing instead of being reinitialized each time.

staticOne.cpp

staticOne.cpp > ...

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

int fun();

int main(){
  cout<<"fun() "<<fun()<<endl;
  cout<<"fun() "<<fun()<<endl;
  cout<<"fun() "<<fun()<<endl;
  cout<<"fun() "<<fun()<<endl;
}

int fun(){
  static int var = 10;
  return var++;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\staticOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
fun() 10
fun() 11
fun() 12
fun() 13
```

19. ***Code :*** Write a program to demonstrate method chaining in C++ by returning *this from a member function, allowing multiple member function calls to be chained together on the same object using the this pointer.

thisFour.cpp

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

class Test{
public:
  Test& funOne(){
    cout<<"in funOne "<<this<<endl;
    return *this;
  }
  void funTwo(){
    cout<<"in funTwo"<<endl;
  }
};

int main(){
  Test objOne;
  /*
  Test& temp = objOne.funOne();//this will return objOne's reference
  temp.funTwo();//using temp's reference used to call funTwo()
  */
  objOne.funOne().funTwo();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\thisFour.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
in funOne 0x61ff1f
in funTwo
```

20. Code : Write a program to demonstrate the use of the this pointer in C++, showing that it holds the address of the calling object, allowing the same member function to distinguish between different object instances at runtime.

thisOne.cpp

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

class Test{
public:
  void funOne(){
    cout<<"in funOne "<<this<<endl;
  }
};

int main(){
  Test objOne;
  Test objTwo;
  cout <<"In main() objOne --> "<<&objOne<<endl;
  cout <<"In main() objTwo --> "<<&objTwo<<endl;
  objOne.funOne();//funOne(&objOne)
  objTwo.funOne();//funOne(&objTwo)
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\thisOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
In main() objOne --> 0x61ff0f
In main() objTwo --> 0x61ff0e
in funOne 0x61ff0f
in funOne 0x61ff0e
```

21. Code : Write a program to demonstrate how the this pointer is used in C++ to differentiate between a class data member and a function parameter with the same name, ensuring the correct assignment to the object's data member.

thisThree.cpp

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

class Test{
  int data;
public:
  void funOne(int data){
    this->data = data;
    cout<<"funOne Data: "<<data<<endl;
  }
  void funTwo(){
    cout<<"funTwo Data: "<<data<<endl;
  }
};

int main(){
  Test obj;
  obj.funOne(10);
  obj.funTwo();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\thisThree.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
funOne Data: 10
funTwo Data: 10
```

22. Code : Write a program to demonstrate how the this pointer refers to the address of the current object, showing that the same member function prints different addresses when invoked by different objects.

thisTwo.cpp

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

class Test{
public:
  void funOne(){
    cout<<"in funOne "<<this<<endl;
  }
};

int main(){
  Test objOne;
  Test objTwo;
  cout <<"In main() objOne --> "<<&objOne<<endl;
  cout <<"In main() objTwo --> "<<&objTwo<<endl;
  objOne.funOne();
  objTwo.funOne();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> g++ .\thisTwo.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_7\Classwork> .\a.exe
In main() objOne --> 0x61ff0f
In main() objTwo --> 0x61ff0e
in funOne 0x61ff0f
in funOne 0x61ff0e
```