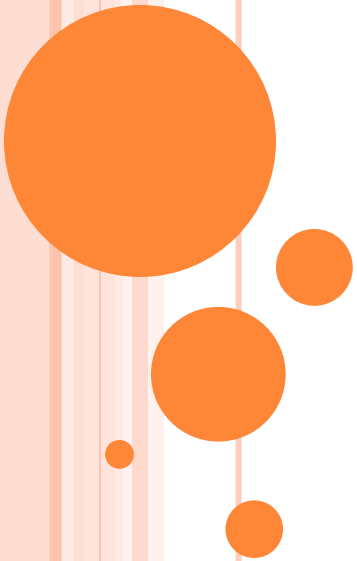


OOP/Computer Programming

Instructor: Dr. Danish Shehzad

Review

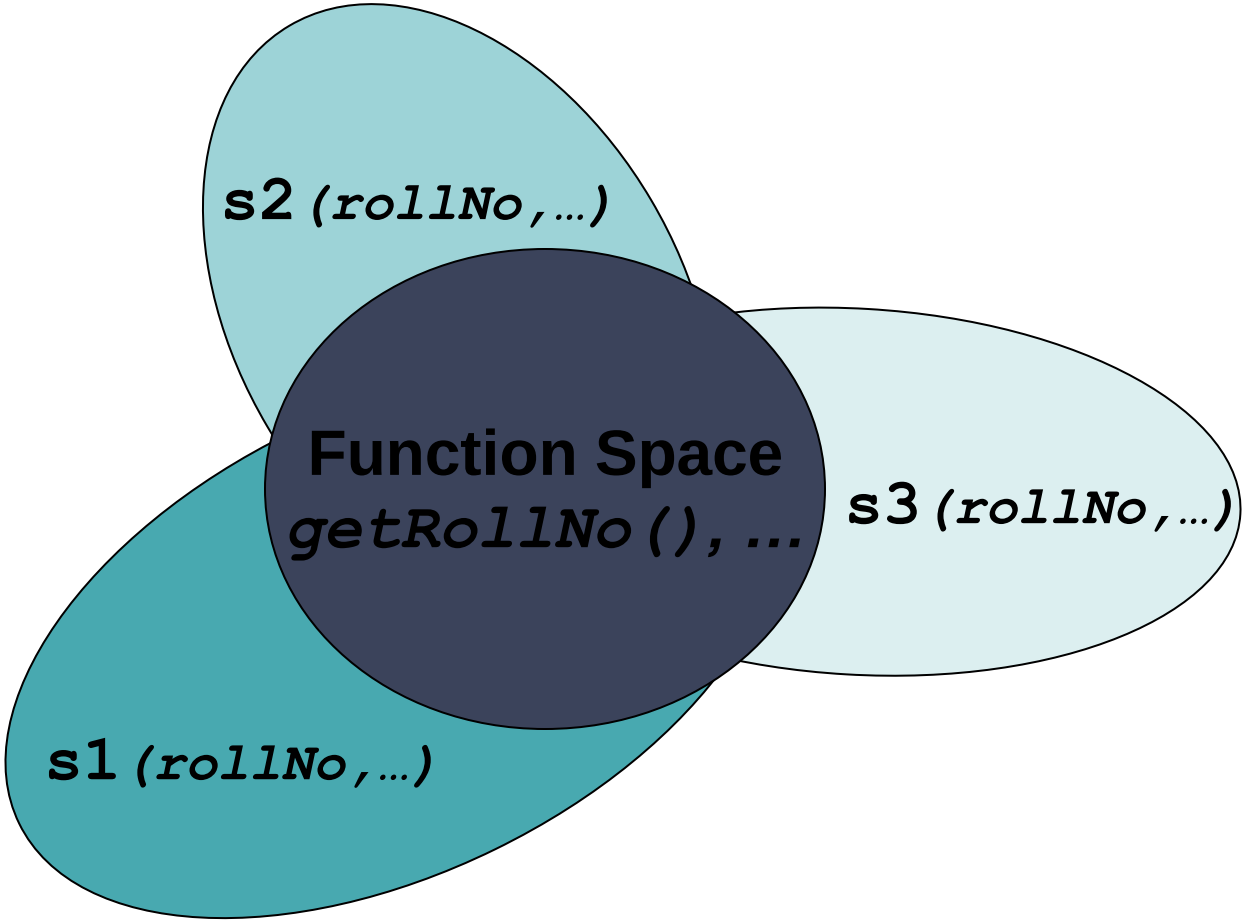
- Class
- Objects
- Functions (Inline, Out of line)
- Interface Vs. Implementation
- Constructors
- Constructor overloading
- Default Constructors
- Copy Constructor
- Shallow Copy vs. Deep Copy
- Destructors

Today's Lecture

- This Pointer
- Constant Functions
- Constant Data Members
- Member Initializer
- Constant Objects

```
class Student{
        int rollNo;
        char *name;
        float GPA;
public:
        int getRollNo();
        void setRollNo(int aRollNo);
…
};
```

- Student *s1, s2, s3;*

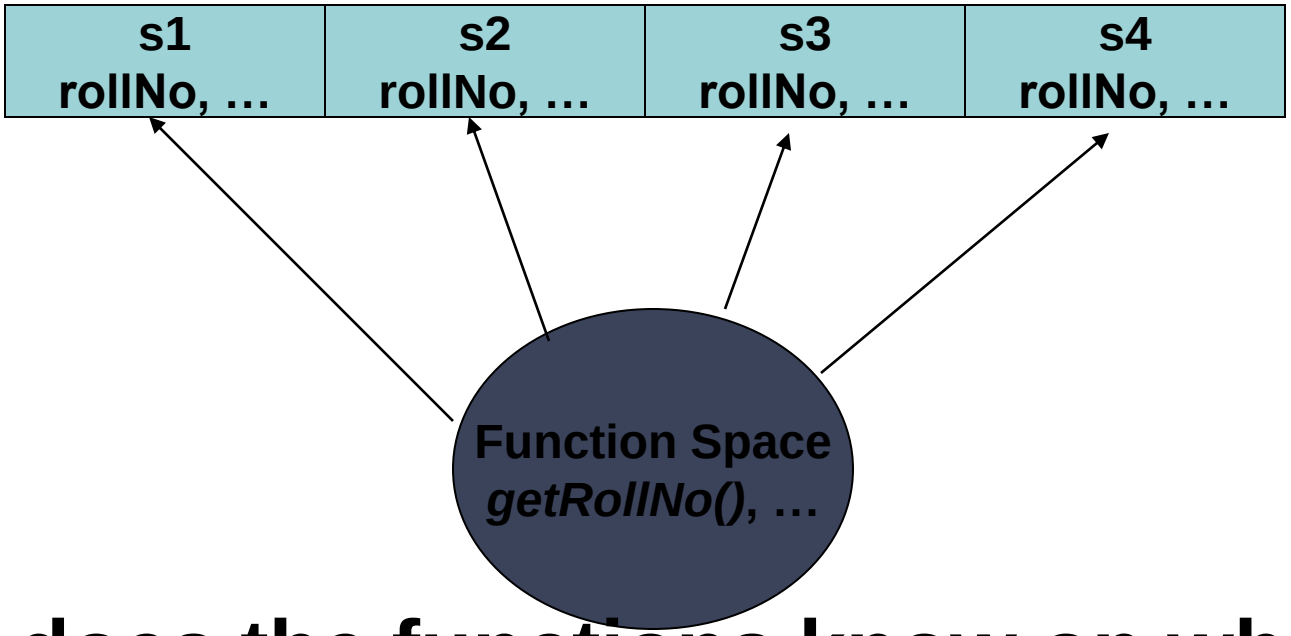


I. this Pointer

- `Student s1, s2, s3;`
- Function space is common for every variable
- Whenever a new object is created:
  - Memory is reserved for variables only
  - Previously defined functions are used over and over again

this Pointer

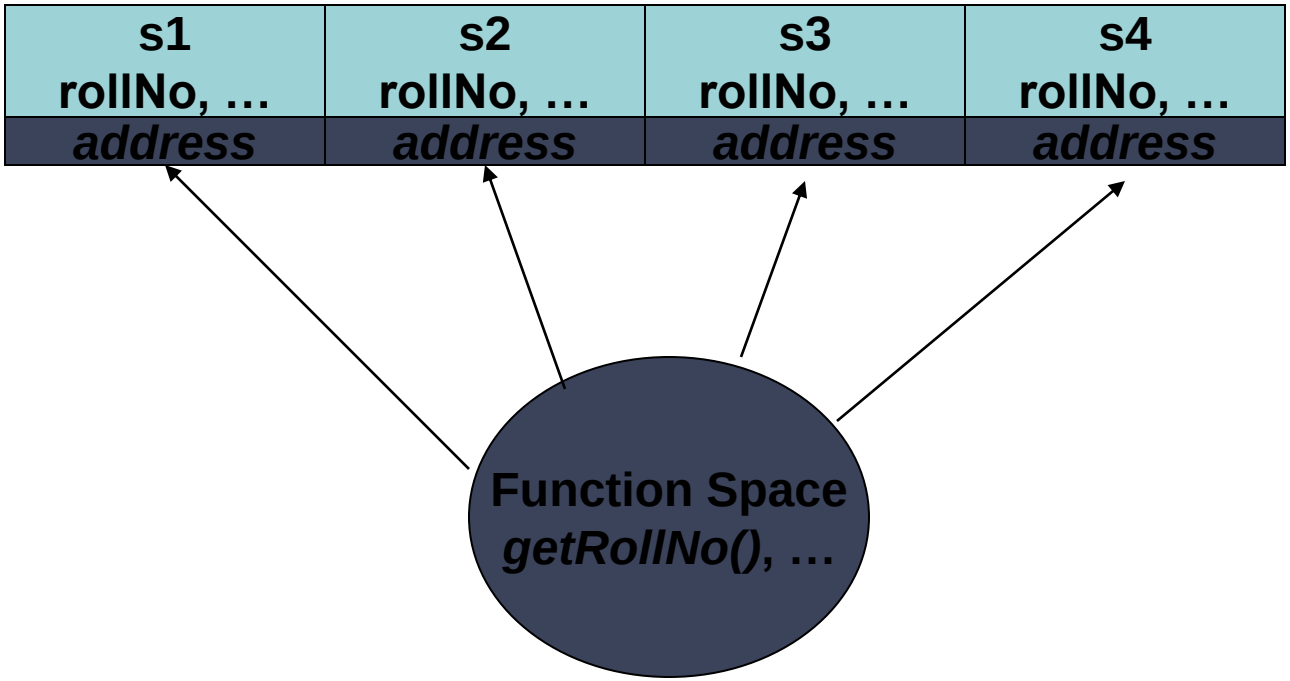
- Memory layout for objects created:



- How does the functions know on which object to act?

THIS POINTER

- Address of each object is passed to the calling function
- This address is deferenced by the functions and hence they act on correct objects



•The variable containing the “self-address” is called this pointer

Passing *this* Pointer

- Whenever a function is called the *this* pointer is passed as a parameter to that function
- Function with *n* parameters is actually called with *n+1* parameters

this Pointer

- There are situations where designer wants to return reference to current object from a function
- In such cases reference is taken from this pointer like (*this)

Example

```
Student& Student::setRollNo(int
  aNo)
{
  …
  return *this;
}
Student& Student::setName(char
  *aName)
{
  …
  return *this;
}
```

Example

```
int main()
{
  Student aStudent;
  Student bStudent;

  bStudent = aStudent.setName("Ahmad");
  …
  bStudent = aStudent.setName("Ali").setRollNo(2);
  return 0;
}
```

Practice 1:

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

class Test
{
private:
int x;
int y;
public:
Test(int c, int d) { x = c; this->y = d; }
Test &setX(int a) { x = a; return *this; }
Test &setY(int b) { y = b; return *this; }
void print() { cout << "x = " << x << " y = " << y << endl; }
};

int main()
{
Test obj1(5, 5);

// Chained function calls. All calls modify the same object
// as the same object is returned by reference
obj1.setX(10).setY(20);

obj1.print();
return 0;
}
```

II. const Member Functions

- There are functions that are meant to be read only
- Keyword **`const`** is placed at the end of the parameter list

const Member Functions

- There must exist a mechanism to detect error if such functions accidentally change the data member
- They are just ***"read-only"***
- Errors due to typing are also caught at compile time

Example

```
class Student{
public:
  int getRollNo() const   {
      return rollNo;
  }
};
```

Example

```
bool Student::isRollNo(int aNo){
  if(rollNo == aNo){
      return true;
  }
  return false;
}
```

Example

```
bool Student::isRollNo(int aNo){
  /*undetected typing mistake*/
  if(rollNo = aNo){
      return true;
  }
  return false;
}
```

Example

```
bool Student::isRollNo(int aNo)const{
  /*compiler error*/
  if(rollNo = aNo){
      return true;
  }
  return false;
}
```

const Functions

- Constructor and destructor are used to modify the object to a well defined state
- Constructors and Destructors cannot be **`const`**

Example

```
class Time{
public:
  Time() const {}
  //error…
  ~Time() const {}
  //error…
};
```

CONST FUNCTION

- Constant member function cannot change data member
- Constant member function cannot access non-constant member functions
- Constructors and Destructors cannot be **`const`**

Example

```
class Student{
  char * name;
public:
  char *getName();
  void setName(char * aName);
  int ConstFunc() const{
      name = getName();//error
      setName("Ahmad");//error
  }
};
```

Practice II.

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

class Test {
        int value;
public:
        Test(int v = 0) {value = v;}
// We get compiler error if we add a line like "value = 100;"
        // in this function.
        int getValue() const {return value;}
};

int main() {
        Test t(20);
        cout<<t.getValue();
        return 0;
}
```

III. Constant Data Members Problem

- Change the class Student such that a student is given a roll number when the object is created and cannot be changed afterwards

Student Class

```
class Student{
…
  int rollNo;
public:
  Student(int aNo);
  int getRollNo();
  void setRollNo(int aNo);
…
};
```

Modified Student Class

```
class Student{
…
  const int rollNo;
public:
  Student(int aNo);
  int getRollNo();
  void setRollNo(int aNo);
…
};
```

Example

```
Student::Student(int aRollNo)
{
  rollNo = aRollNo;
  /*error: cannot modify a
  constant data member*/
}
```

Example

```
void Student::SetRollNo(int i)
{
  rollNo = i;
  /*error: cannot modify a constant data
  member*/
}
```

IV.
Member Initializer List

- A member initializer list is a mechanism to initialize data members
- It is given after closing parenthesis of parameter list of constructor
- In case of more then one member use comma separated list

Example

```
class Student{
  const int rollNo;
  char *name;
  float GPA;
public:
  Student(int aRollNo):rollNo(aRollNo),name(Null),
  GPA(0.0){
       …
  }
…
};
```

Order of Initialization

- Data member are initialized in order they are declared
- Order in member initializer list is not significant at all

Example

```
class ABC{
  int x;
  int y;
  int z;
public:
  ABC();
};
```

Example

```
ABC::ABC():y(10),x(y),z(y)
{
  …
}
/*  x = Junk value
    y = 10
    z = 10 */
```

Practice III.

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

class Test {
        const int t;
public:
        Test(int t):t(t) {} //Initializer list must be used
        int getT() { return t; }
};

int main() {
        Test t1(10);
        cout<<t1.getT();
        return 0;
}

/* OUTPUT:
10
*/
```

https://www.geeksforgeeks.org/when-do-we-use-initializer-list-in-c/

V.

CONST OBJECTS

- Objects can be declared constant with the use of const keyword
- Constant objects cannot change their state

Example

```
int main()
{
  const Student aStudent;
  return 0;
}
```

Example

```
class Student{
…
  int rollNo;
public:
…
  int getRollNo(){
      return rollNo;
  }
};
//error
```

Example

```
int main(){
  const Student aStudent;
  int a = aStudent.getRollNo();
  //error
}
```

const Objects

- **`const`** objects cannot access "*non const*" member function
- Chances of unintentional modification are eliminated

Example

```
class Student{
…
  int rollNo;
public:
…
  int getRollNo()const{
      return rollNo;
  }
};
```