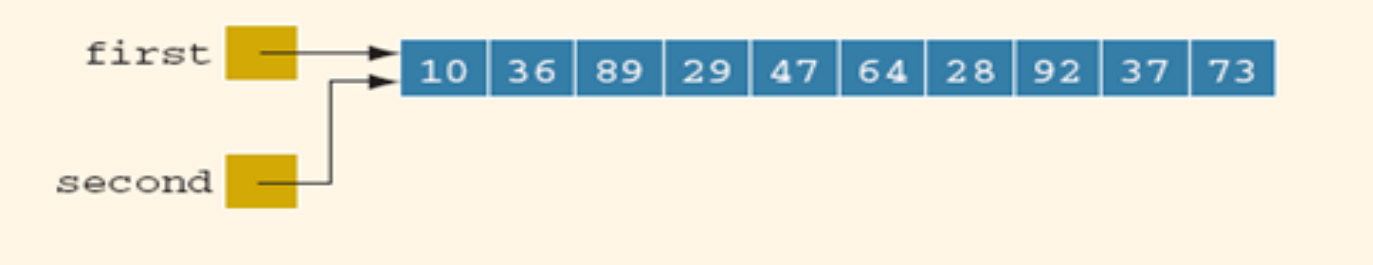
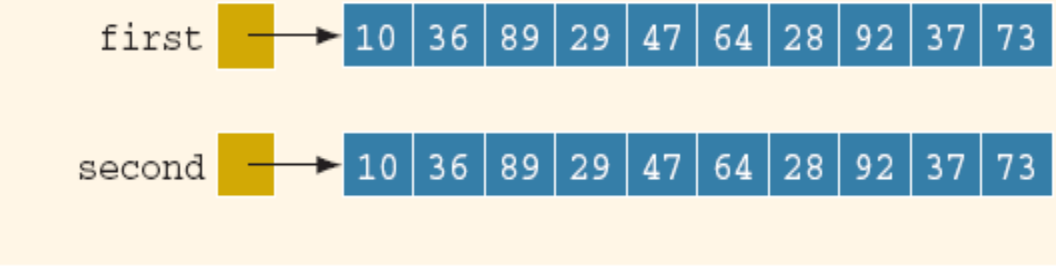
Week 04

- Copy Constructor
- Shallow vs Deep Copy
- Private Functions
- Constant Data & Functions

Shallow Vs Deep Copy



first and second after the statement second = first; executes



first and second both pointing to their own data

Copy Constructor

Copy Constructor

- Copy constructor are used when:
  - Initializing an object at the time of creation
  - When an object is passed by value to a function

Example

```
void func1(Student student){
…
}
int main(){
  Student studentA(“Ahmad”);
  Student studentB = studentA;
  func1(studentA);
}
```

Copy Constructor (contd.)

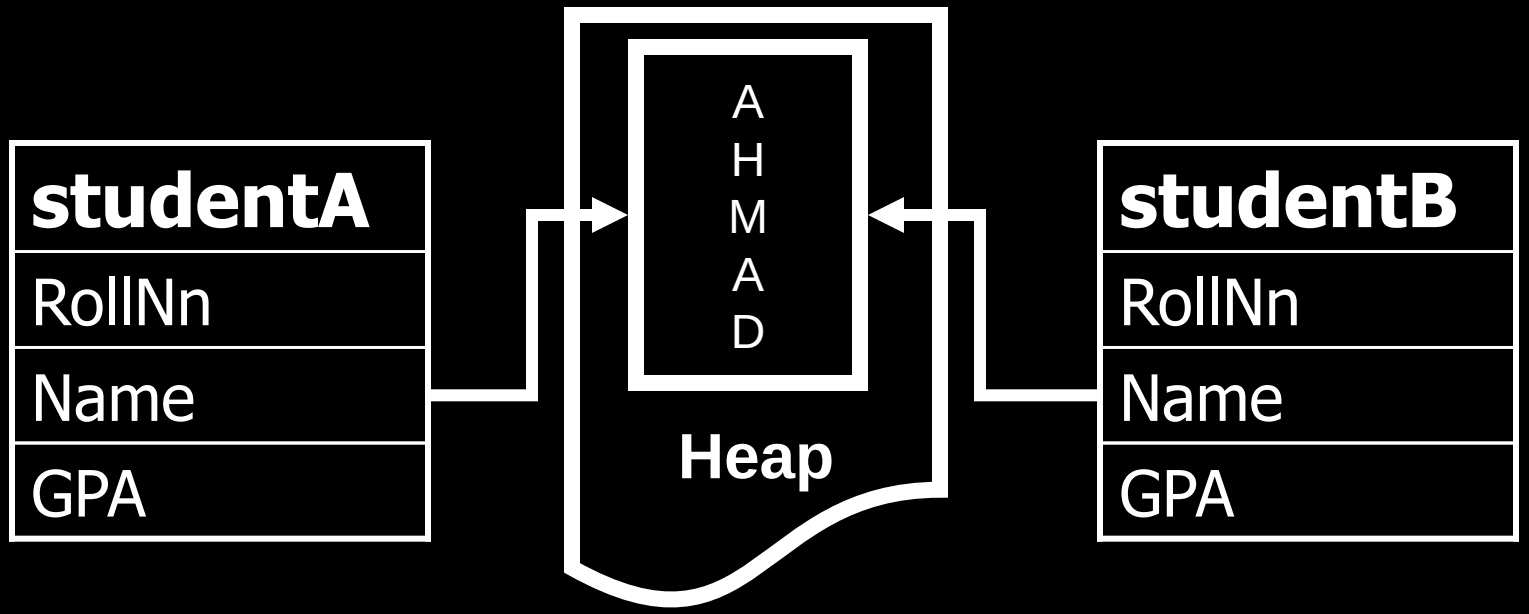
```
//compiler generated copy
  constructor
Student::Student(
    const Student &obj){
  rollNo = obj.rollNo;
  name = obj.name;
  GPA = obj.GPA;
}
//member-wise copy
```

Shallow Copy

- When we initialize one object with another then the compiler copies state of one object to the other
- This kind of copying is called shallow copying

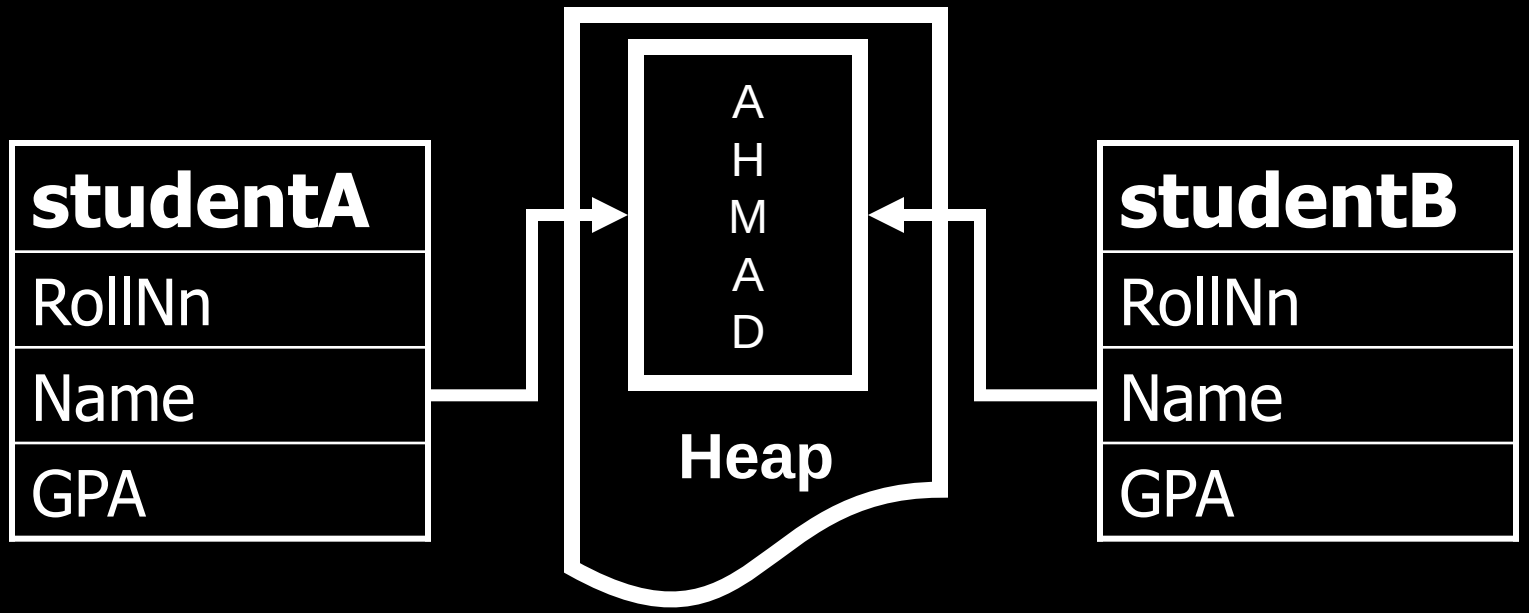
Example

```
Student studentA("Ahmad");
Student studentB = studentA;
```



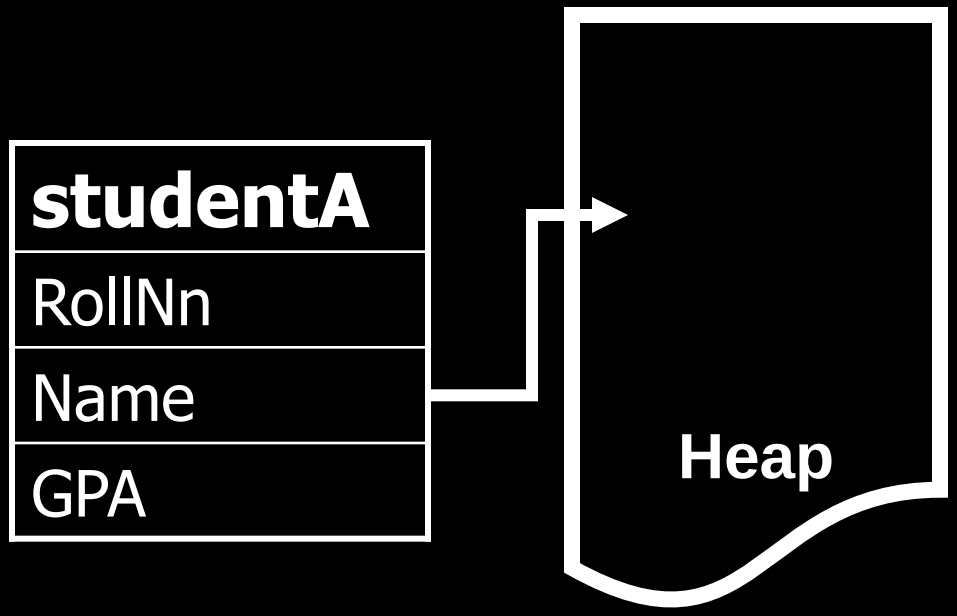
Example

```
int main(){
 Student studentA("Ahmad",1);
 {
  Student studentB = studentA;
```



Example

```
int main(){
 Student studentA(“Ahmad”,1);
 {
  Student studentB = studentA;
 }
}
```

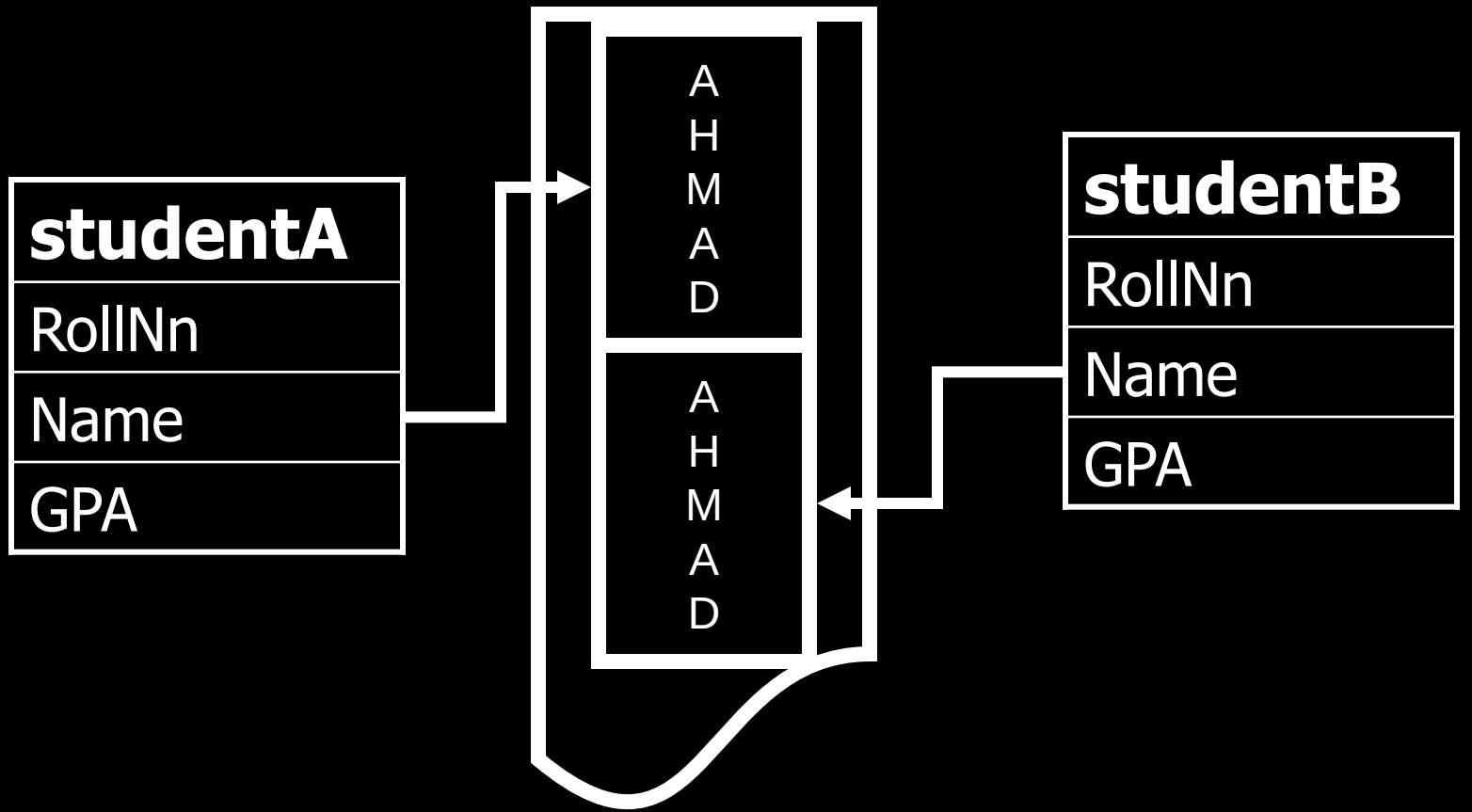


Copy Constructor (contd.)

```
//user defined copy constructor
Student::Student(
          const Student &   obj){
  int len = strlen(obj.name);
  name = new char[len+1]
  strcpy(name, obj.name); //deep copy
  …
  /*copy rest of the data members*/
}
```

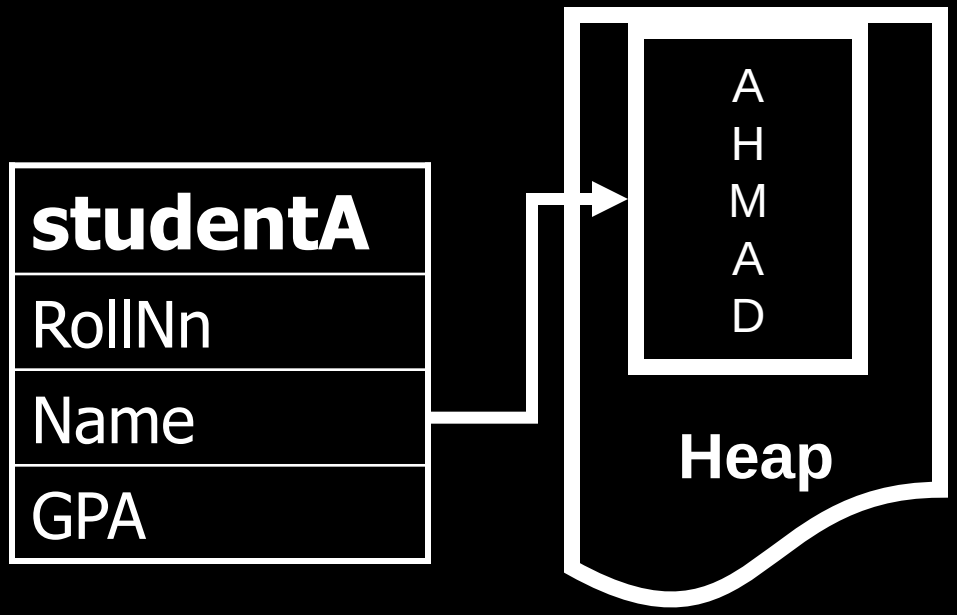
Example

```
int main(){
 Student studentA("Ahmad",1);
 {
  Student studentB = studentA;
```



Example

```
int main(){
 Student studentA(“Ahmad”,1);
 {
  Student studentB = studentA;
 }
}
```



Copy Constructor (contd.)

- Copy constructor is normally used to perform deep copy
- If we do not make a copy constructor then the compiler performs shallow copy

Destructor

- Destructor is used to free memory that is allocated through dynamic allocation
- Destructor is used to perform house keeping operations

Destructor (contd.)

- Destructor is a function with the same name as that of class, but preceded with a tilde '~'

Example

```
class Student
{
  …
public:
  ~Student(){
     if(name){
         delete []name;
     }
  }
}
```

Overloading

- Destructors cannot be overloaded

Sequence of Calls

- Constructors and destructors are called automatically
- Constructors are called in the sequence in which object is declared
- Destructors are called in reverse order

Example

```
Student::Student(char * aName){
  …
     cout << aName << “Cons\n”;
  }
  Student::~Student(){
     cout << name << “Dest\n”;
  }
};
```

Example

```
int main()
{
  Student studentB(“Ali”);
  Student studentA(“Ahmad”);
  return 0;
}
```

Example

Output:

Ali Cons

Ahmad Cons

Ahmad Dest

Ali Dest

Lecture 12

this Pointer

Constant Functions and Data

Private Functions/ Utility Functions/ Helper Functions

- A utility function is not part of a class's public interface; rather, it's a private member function that supports the operation of the class's other member functions.

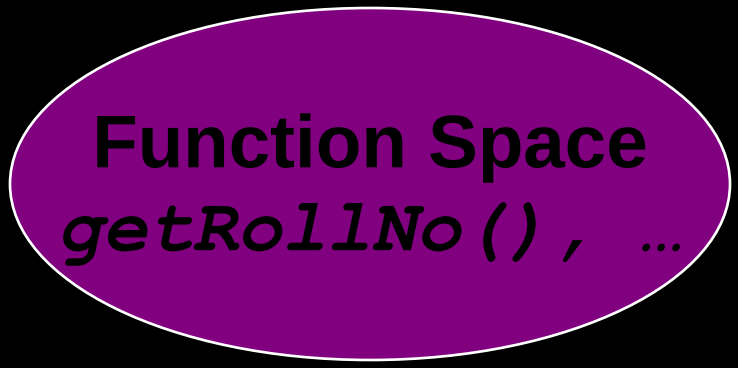
- Utility functions are not intended to be used by clients of a class (but can be used by friends of a class).

this Pointer

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

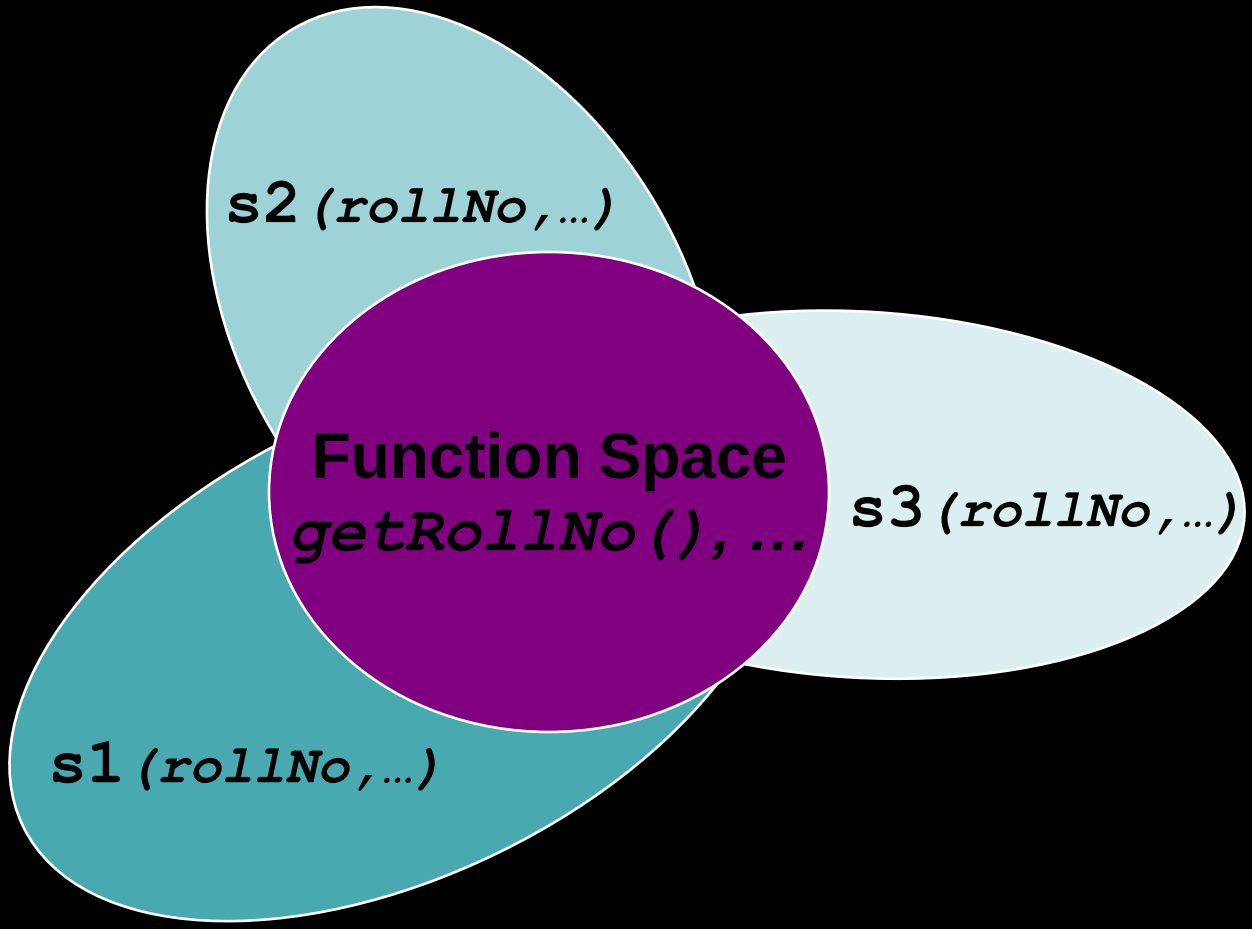
this Pointer

- The compiler reserves space for the functions defined in the class
- Space for data is not allocated (*since no object is yet created*)



this Pointer

- `Student s1, s2, s3;`

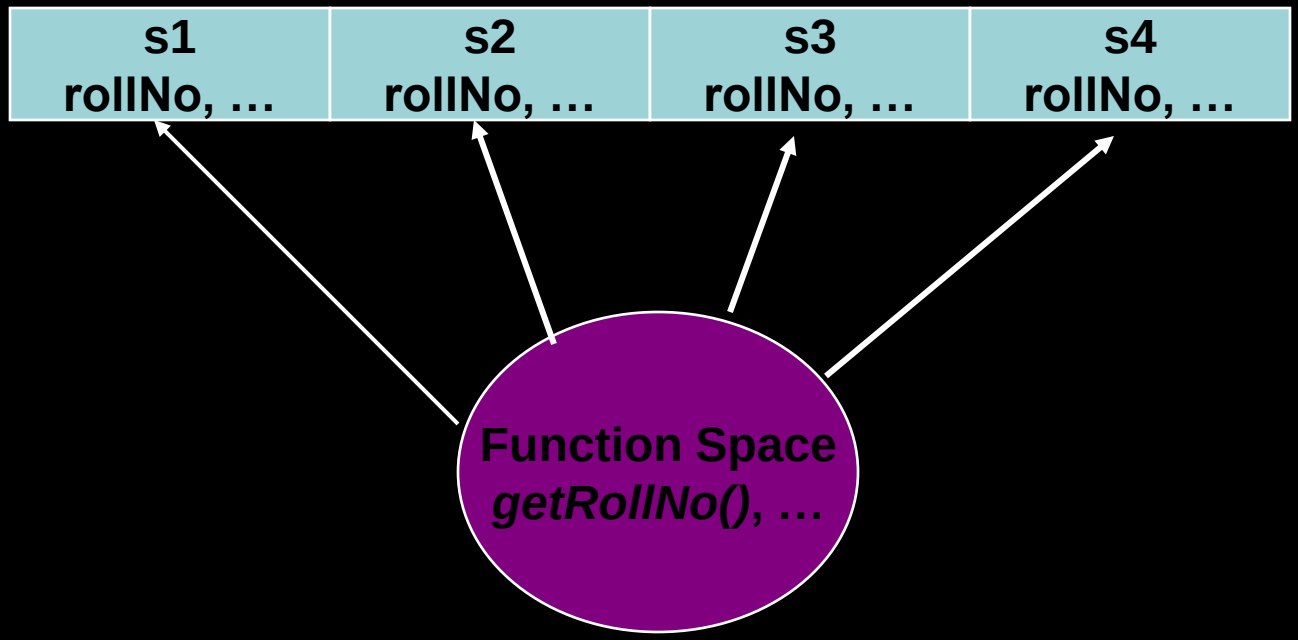


this Pointer

- 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

s1 rollNo, …	s2 rollNo, …	s3 rollNo, …	s4 rollNo, …
address	*address*	*address*	*address*

•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

Example

```
void Student::setName(char *)
```

is internally represented as

```
void Student::setName(char *,
  const Student *)
```

Declaration of this

```
DataType * const this;
```

Compiler Generated Code

```
Student::Student(){
  rollNo = 0;
}

Student::Student(){
  this->rollNo = 0;
}
```

Review