Object Oriented Programming (OOP)

Review

- Class
  - Concept
  - Definition
- Data members
- Member Functions
- Access specifier

Member Functions

- Member functions are the functions that operate on the data encapsulated in the class
- Public member functions are the interface to the class

Member Functions (contd.)

- Define member function inside the class definition

  OR

- Define member function outside the class definition
  - But they must be declared inside class definition

Function Inside Class Body

```
class ClassName {
    ...
    public:
    ReturnType FunctionName() {
        ...
    }
};
```

Example

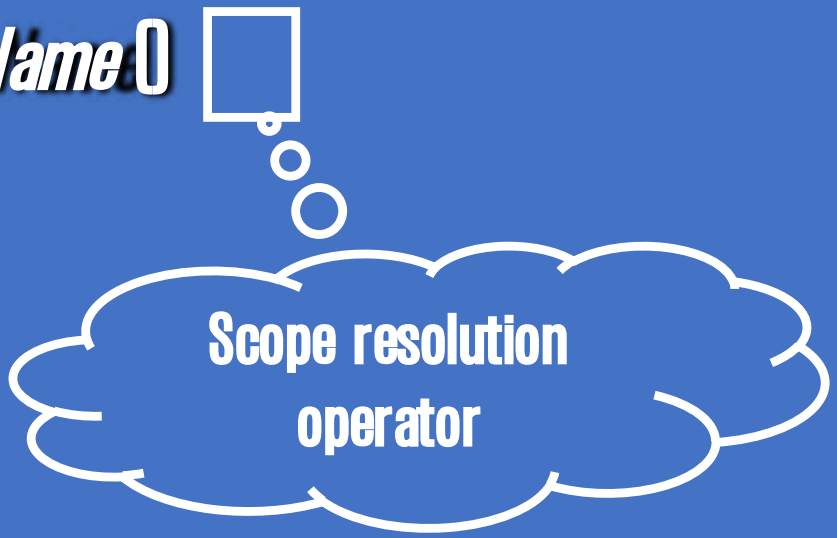
- Define a class of student that has a roll number. This class should have a function that can be used to set the roll number

Example

```
class Student{
    int rollNo;
public:
    void setRollNo(int aRollNo){
        rollNo = aRollNo;
    }
};
```

Function Outside Class Body

```
class ClassName {
    ...
    public:
    ReturnType FunctionName ();
};
ReturnType ClassName :: FunctionName ()
{
    ...
}
```



Example

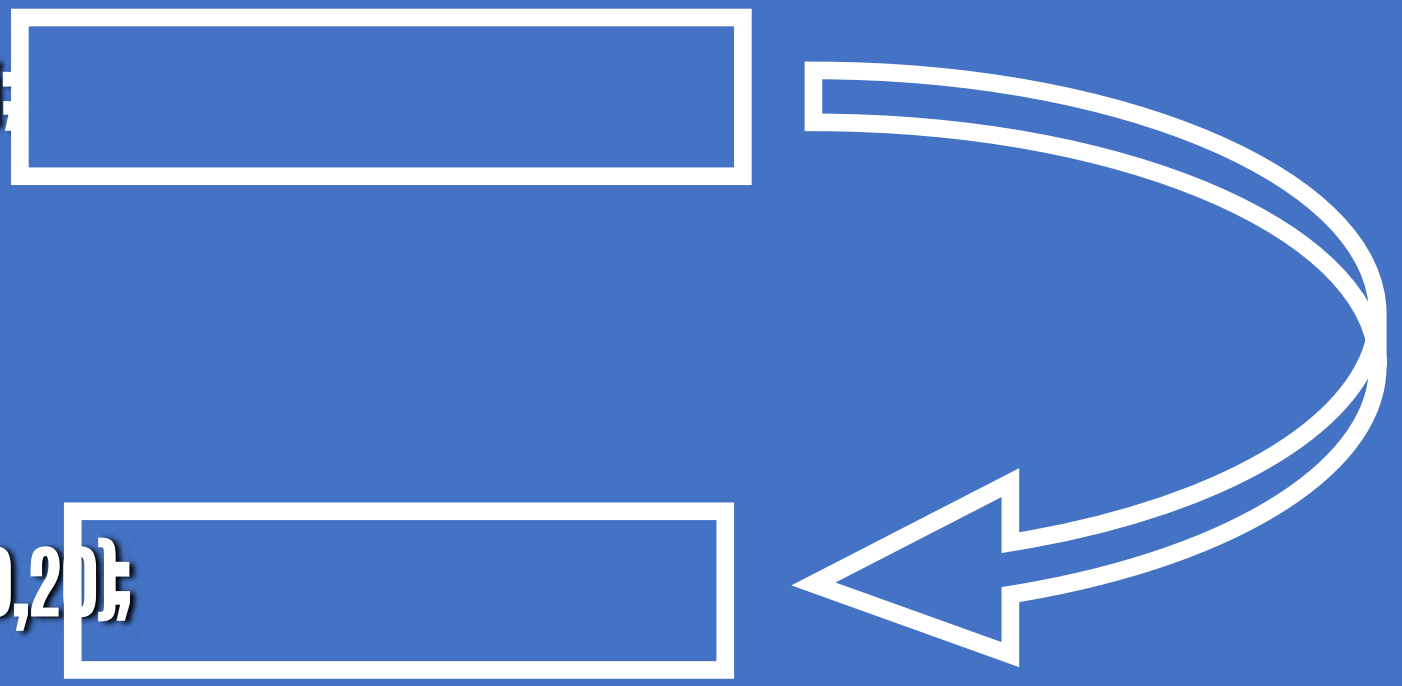
```
class Student{
    ...
    int rollNo;
public:
    void setRollNo(int aRollNo);
};
void Student::setRollNo(int aRollNo){
    ...
    rollNo = aRollNo;
}
```

Inline Functions

- Instead of calling an inline function compiler replaces the code at the function call point
- Keyword ‘inline’ is used to request compiler to make a function inline
- It is a request and not a command

Example

```
inline int Area(int len, int hi)
{
   return len * hi;
}
int main()
{
   cout << Area(10,20);
}
```



Inline Functions

- If we define the function inside the class body then the function is by default an inline function
- In case function is defined outside the class body then we must use the keyword ‘inline’ to make a function inline

Example

```
class Student{
    int rollNo;
public:
    void setRollNo(int aRollNo){
        ...
        rollNo = aRollNo;
    }
};
```

Example

```
class Student{
    ...
    public:
    inline void setRollNo(int aRollNo);
};
void Student::setRollNo(int aRollNo){
    ...
    rollNo = aRollNo;
}
```

Example

```
class Student{
    ...
    public:
    void setRollNo(int aRollNo);
};
inline void Student::setRollNo(int                    aRollNo){
    ...
    rollNo = aRollNo;
}
```

Example

```
class Student{
    ...
    public:
    inline void setRollNo(int aRollNo);
};
inline void Student::setRollNo(int                    aRollNo){
    ...
    rollNo = aRollNo;
}
```

Constructor

Constructor

- Constructor is used to initialize the objects of a class
- Constructor is used to ensure that object is in well defined state at the time of creation
- Constructor is automatically called when the object is created
- Constructor are not usually called explicitly

Constructor (contd.)

- Constructor is a special function having same name as the class name
- Constructor does not have return type
- Constructors are commonly public members

Example

```
class Student{
    ...
public:
    Student(){
        rollNo = 0;
        ...
    }
};
```

Example

```
int main()
{
   Student aStudent;
   /*constructor is implicitly called at this point*/
}
```

Default Constructor

- Constructor without any argument is called default constructor
- If we do not define a default constructor the compiler will generate a default constructor
- This compiler generated default constructor initialize the data members to their default values

Example

```
class Student
{
   int rollNo;
   char *name;
   float GPA;
public:
   ...      //no constructors
};
```

Example

```
Compiler generated default constructor
{
   rollNo = 0;
   GPA = 0.0;
   name = NULL;
}
```

Constructor Overloading

- Constructors can have parameters
- These parameters are used to initialize the data members with user supplied data

Example

```
class Student{
...
public:
    Student();
    Student(char * aName);
    Student(char * aName, int aRollNo);
    Student(int aRollNo, int aRollNo, float aGPA);
};
```

Example

```
Student::Student(int aRollNo,
                 char * aName){
    if(aRollNo < 0){
        rollNo = 0;
    }
    else {
    rollNo = aRollNo;
    }
    ...
}
```

Example

```
int main()
{
    Student student1;
    Student student2("Name");
    Student student3("Name", 1);
    Student student4("Name",1,4.0);
}
```

Constructor Overloading

- Use default parameter value to reduce the writing effort

Example

```
Student::Student(  char * aName = NULL,
                   int aRollNo= 0,
                   float aGPA = 0.0){
    ...
}
```

Is equivalent to

```
Student();
Student(char * aName);
Student(char * aName, int aRollNo);
Student(char * Name, int aRollNo, float aGPA);
```

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;
    Student studentB = studentA;
    func1(studentA);
}
```

Copy Constructor (Syntax)

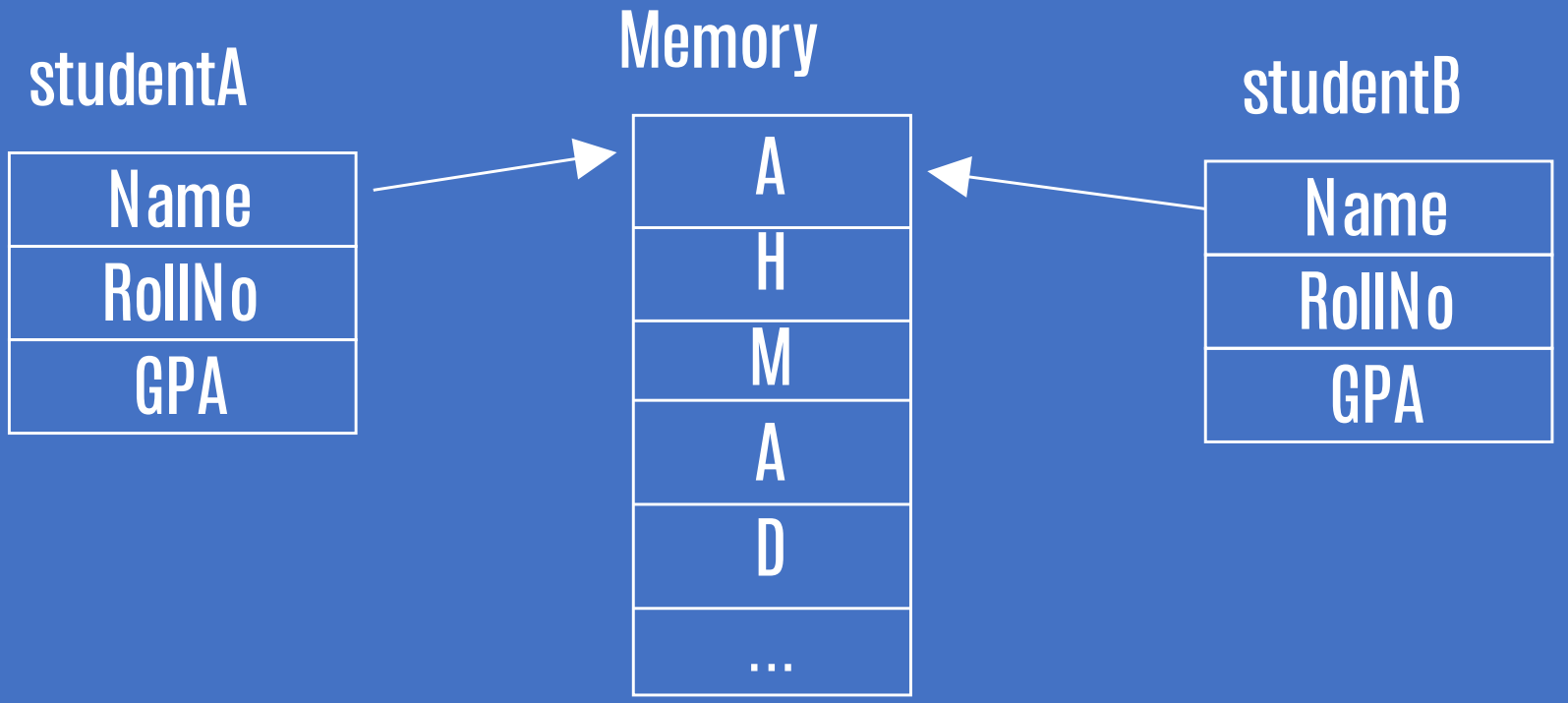
```
Student::Student(
        const Student &obj){
    rollNo = obj.rollNo;
    name = obj.name;
    GPA = obj.GPA;
}
```

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;
Student studentB = studentA;
```



Copy Constructor (contd.)

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

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

Example

```
Student studentA;
Student studentB = studentA;
```