Object oriented programming (OOP)

Class

- Class is a tool to realize objects
- Class is a tool for defining a new type

Example

- Lion is an object
- Student is an object
- Both has some attributes and some behaviors

Uses

- The problem becomes easy to understand
- Interactions can be easily modeled

Type in C++

- Mechanism for user defined types are
  - Structures
  - Classes
- Built-in types are like int, float and double
- User defined type can be
  - Student in student management system
  - Circle in a drawing software

Abstraction

- Only include details in the system that are required for making a functional system
- Student
  - Name
  - Address
  - Sibling
  - Father Business

Relevant to our problem

Not relevant to our problem

Defining a New User Defined Type

```
class ClassName
{
    ...
    DataType MemberVariable;
    ReturnType MemberFunction();
    ...
};
```

Syntax

Syntax

Example

```
class Student
{
    int rollNo;
    char *name;
    float CGPA;
    char *address;
...
    void setName(char *newName);
    void setRollNo(int newRollNo);
...
};
```

Member variables

Member Functions

Why Member Function

- They model the behaviors of an object
- Objects can make their data invisible
- Object remains in consistent state

Example

```
Student aStudent;

aStudent.rollNo = 514;

aStudent.rollNo = -514; //Error
```

Object and Class

- Object is an instantiation of a user defined type or a class

Declaring class variables

- Variables of classes (objects) are declared just like variables of structures and built-in data types

```
TypeName VaraibaleName;
int var;
Student aStudent;
```

Accessing members

- Members of an object can be accessed using
  - dot operator (.) to access via the variable name
  - arrow operator (->) to access via a pointer to an object
- Member variables and member functions are accessed in a similar fashion

Example

```
class Student{
    int rollNo;
    void setRollNo(int aNo);
};

    Student aStudent;
    aStudent.rollNo;
```

Access specifiers

Access specifiers

- There are three access specifiers
  - ‘public’ is used to tell that member can be accessed whenever you have access to the object
  - ‘private’ is used to tell that member can only be accessed from a member function
  - ‘protected’ to be discussed when we cover inheritance

Example

```
class Student{
private:
    char * name;
    int rollNo;
public:
    void setName(char *);
    void setRollNo(int);
...
};
```

Cannot be accessed outside class

Can be accessed outside class

Example

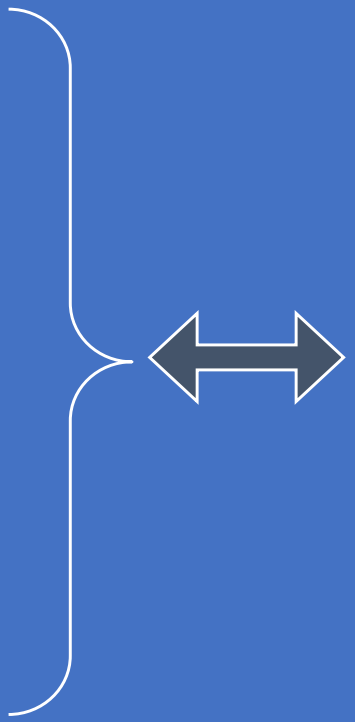
```
class Student{
...
    int rollNo;
public:
    void setRollNo(int aNo);
};
int main(){
    Student aStudent;
    aStudent.SetRollNo(1);
}
```

Default access specifiers

- When no access specifier is mentioned then by default the member is considered private member

Example

```
class Student
{
    char * name;
    int RollNo;
};
```



```
class Student
{
private:
    char * name;
    int RollNo;
};
```

Example

```
class Student
{
   char * name;
   int RollNo;
   void SetName(char *);
};
Student aStudent;
aStudent.SetName(Ali);
```

Example

```
class Student
{
    char * name;
    int RollNo;
public:
    void setName(char *);
};
Student aStudent;
aStudent.SetName("Ali");
```