

Member functions

Lecture-3

Scope

- Access functions
- Utility functions
- Constructors
- How to prevent multiple inclusions of header file?

Access and utility functions

- Access functions - *public* functions that usually read or display the data(members)
 - predicate functions – to test truth or falsity of conditions like *isPositive()*
- Utility functions – *private* functions that support the operations of other member functions

Example

```
class salesperson
```

```
{
```

```
    public:
```

```
        void getsales();
```

```
        void setsales();
```

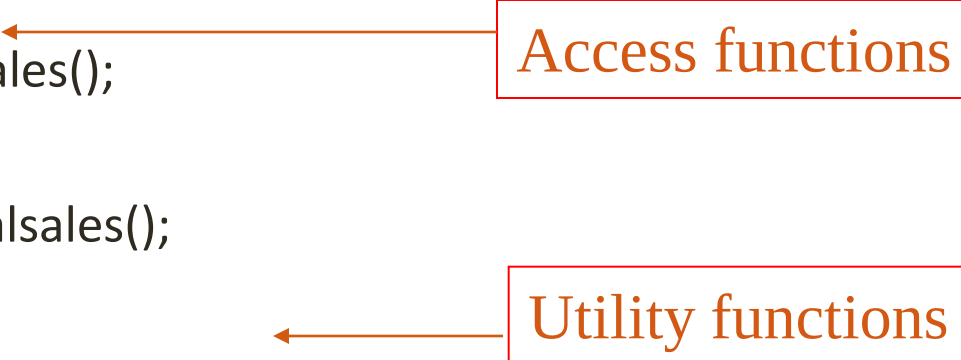
```
        void printannualsales();
```

```
    private:
```

```
        double totalannualsales();
```

```
        double sales[12];
```

```
};
```



Access functions

Utility functions

Initialization of data members

- Can we initialize the data members of the objects, at the time of its creation?

```
class employee
```

```
{
```

```
private: double wage; char name[80];
```

```
public: void putwage(double w); double getwage();
```

```
void putname(char *); void getname(char *) ;
```

```
};
```

```
int main() { employee ted; }
```



Object creation

Constructors

- A constructor is a special function
 - that must be defined with the same name as the class
 - The constructor call occurs implicitly when the object is created
 - We can write initialization code for data members in the constructor
 - Should be declared as public members

Constructors (contd..)

- Cannot return values
- C++ requires a constructor call for each object that is created, in any class that does not explicitly include a constructor, the compiler provides a default constructor – a constructor with no arguments
- Notes – *if a constructor with arguments is specified, c++ will not implicitly create a default constructor*

Example

```
• class stack {  
    private:  
        int stck[100];  
        int top;  
    public:  
        stack();  
        void push(int i);  
        int pop();  
};
```

```
stack::stack()  
{ top=0;  
  cout<<"initialized"; }
```

```
int main()  
{  
    stack s1;  
}
```

Parameterized constructors

- To initialize the data members through constructors, we can use parameters in the constructors

```
■ class stack {  
    private: int stck[100];  
           int top;  
    public: stack(int i);  
           void push(int i);  
           int pop();  
};
```

```
stack::stack(i)  
{ top=i; cout<<"initialized"; }
```

```
int main()  
{ stack s1(1); }
```

Parameterized constructors (contd...)

- Initial values should be passed as arguments. The object creation can take either of the two forms
 - `stack s1=stack(1)`
 - `stack s1(1);`

Default arguments

- C++ allows us to call a function without specifying all its arguments
- Function assigns a default value to the parameter which does not have a matching argument in function call
- Specified when the function is declared

float amount(float principal, int period, float rate=0.15)

Constructors with default arguments

- It is possible to define constructors with default arguments.

```
■ class stack {  
    private: int stck[100];  
            int top;  
    public: stack(int i=0);  
            void push(int i);  
            int pop();  
};
```

```
stack::stack(int i=0)  
{ top=i; cout<<"initialized"; }
```

```
int main()  
{ stack s1(1); stack s2; }
```

Preventing multiple inclusion of header file

- Preprocessor wrapper - `#ifndef .... #endif`

```
#ifndef EMPLOYEE_H
```

```
#define EMPLOYEE_H
```

```
class employee
```

```
{ // class begins
```

```
char name[80];
```

```
public: void putname(char *); void getname(char *) ;
```

```
private: double wage;
```

```
public: void putwage(double w); double getwage();
```

```
}; // class ends here
```

```
#endif
```

employee.h

Class Assignment

- What is the difference between a local variable and a data member?
- Find errors –

```
class time {  
    public : // some functions are declared  
    private : int hour=0; int min=0; int second=0;  
}
```