COURSE CODE: CSE 205
COURSE TITLE: OBJECT ORIENTED PROGRAMMING

Prepared by- ***Sumaiya Afroz Mila***

Operations in class

- Assigning objects
- Passing object to function
- Returning object from function
- Array of objects
- Create pointer/ reference of objects

Assigning Objects

- One object can be assigned to another if both objects are of the same type (both are objects of the same class)
- When one object assigned to another, bitwise copy of all the members is made.

Assigning Objects Example

```
class student{
    char *name;
    int id;
public:
    void setValues(char *n, int i){
        name = n;
        id = i;
    }
    void showValues(){
        cout<<"Name is :"<<name<<endl;
        cout<<"ID is: "<<id<<endl;
    }
};
```

```
int main(){
    student ob1,ob2;
    ob1.setValues("mila", 21);
    ob1.showValues();
```

What will be the output??

F:\programming\Untitled1.exe

```
Name is :mila
ID is: 21

Process returned 0 (0x0)   execution time : 0.162 s
Press any key to continue.
```

Assigning Objects Example

```
class student{
    char *name;
    int id;
public:
    void setValues(char *n, int i){
        name = n;
        id = i;
    }
    void showValues(){
        cout<<"Name is :"<<name<<endl;
        cout<<"ID is: "<<id<<endl;
    }
};
```

OUTPUT???

```
int main(){
    student ob1,ob2;
    ob1.setValues("mila", 21);
    ob1.showValues();

    ob2 = ob1;
cout<<"after assigning ob1 to ob2";
    ob2.showValues();
}
```

```
Name is :mila
ID is: 21

after assigning ob1 to ob2
Name is :mila
ID is: 21
```

Assigning Objects Example

```
class student{
    char *name;
public:
    void setValues(char *n){
        name = n;
    }
    void showValues(){
        cout<<"Name is :"<<name;
        cout<<endl;
    }
};
```

```
class teacher{
    char *name;
public:
    void setValues(char *n){
        name = n;
    }
    void showValues(){
        cout<<"Name is :"<<name;
        cout<<endl;
    }
};
```

```
int main(){
    student ob1;
    teacher ob2;
    ob1.setValues("mila");
    ob1.showValues();
}
```

Output??

Name is: mila

Assigning Objects Example

```
class student{
    char *name;
public:
    void setValues(char *n){
        name = n;
    }
    void showValues(){
        cout<<"Name is :"<<name;
        cout<<endl;
    }
};
```

```
class teacher{
    char *name;
public:
    void setValues(char *n){
        name = n;
    }
    void showValues(){
        cout<<"Name is :"<<name;
        cout<<endl;
    }
};
```

ERROR
Assignment not allowed as objects are not of the same class

```
int main(){
    student ob1;
    teacher ob2;
    ob1.setValues("mila");
    ob1.showValues();

    ob2 = ob1;
cout<<endl;
cout<<"after assigning ob1 to ob2";
cout<<endl;
    ob2.showValues();
}
```

Passing Objects to a function

- Objects can be passed to functions as argument in the same way other types of data are passed.
- Simply declare the functions parameter as a class type
- Use an object of that class as an argument when calling the function
- As with other types of data, by default all objects are passed by value to a function.
  - Means a bitwise copy of the argument is made
  - This copy is used by the function.
  - Changes to the object inside the function do not affect the calling object.

Syntax

```
void func(student ob){
 // function body
}

int main(){
Student ob;
func(ob);
}
```

Passing Objects to a function Example

- Design a Point class which will contain the variables X, Y(private)
- Create a function "SetXY(int a, int b)" to set the value of X and Y.
- Create an object p1 of the class. Set the values.
- Create a function "Distance (Point p2)" which will calculate and show the distance between two points p1 and p2.
- Formula to calculate distance: d = $\sqrt{(X_2 - X)^2 + (Y_2 - Y)^2}$
- Include the "math.h" to use sqrt() function

```
class Point{
private:
    int X;
    int Y;
public:
    void setXY(int a, int b);
    void Distance(Point p2);
};
int main(){
    Point p1,p2;
    p1.setXY(0,0);
    p2.setXY(4,0);
    p1.Distance(p2);
}
```

Returning Objects from a function

- Objects can be returned from functions in the same way other types of data are returned.
- Simply declare the function as returning a class type
- Return an object from that type using return statement
- When an object is returned by a func, a temporary object is automatically created which holds the return value.
- After returning the value, this object is destroyed

Syntax

```
student func(){
        student ob1;
        return ob1;
}
int main(){
Student ob2;
ob2 = func();
}
```

Returning Objects from a function Example

- Design a Point class which will contain the variables X, Y(private)
- Create a function “SetXY(int a, int b)” to set the value of X and Y.
- Create a “showXY()” function to show the values of the coordinates
- Create an object p1 of the class. Set the values.
- Create a function “CalcMidPoint(Point p2)” which will calculate and return the midpoint between two points p1 and p2.
- Formula to calculate midpoint:
  - x = (x1+x2)/2 , y = (y1+y2)/2

```
class Point{
private:
    int X;
    int Y;
public:
    void setXY(float a,float b);
    Point CalcMidPoint(Point p2);
    void showXY();
};
int main(){
    Point p1,p2,p3;
    p1.setXY(0,0);
    p2.setXY(6,4);
    p3 = p1.CalcMidPoint(p2);
    p3.showXY();
}
```

Array of Objects

- Objects can be arrayed.
- Syntax for declaring an array of objects is exactly same as declaring an array of other type of variables
- Arrays of objects are accessed in the same way as accessing arrays of other types of variables

Syntax

Declaration:

```
Class_name  object_name[size];
```

Accessing:

```
object_name[index].member_variable;
object_name[index].member_function();
```

Rewrite this Code declaring array of objects

```
class student{
    char *name;
    int id;
public:
    void setValues();
    void showValues();
};
void student::setValues(){
    name = (char *)malloc(10*sizeof(char));
    cin>>name;
    cin>>id;
}
void student::showValues(){
    cout<<"Name is: "<<name<<endl;
    cout<<"ID is: "<<id<<endl;
}
```

```
int main(){
        student ob1,ob2;

        ob1.setValues();
        ob2.setValues();

        ob1.showValues();
        ob2.showValues();
}
```

Rewrite this Code declaring array of objects

```
class student{
    char *name;
    int id;
public:
    void setValues();
    void showValues();
};
void student::setValues(){
    name = (char *)malloc(10*sizeof(char));
    cin>>name;
    cin>>id;
}
void student::showValues(){
    cout<<"Name is: "<<name<<endl;
    cout<<"ID is: "<<id<<endl;
}
```

```
int main(){
        student ob[2];

        for(int i=0; i<2;i++){
            ob[i].setValues();
        }
        for(int i=0; i<2;i++){
            ob[i].showValues();
        }
}
```

Using pointer to objects

- Objects can be accessed via pointers.
- When a pointer to an object is used, the **object's members** are referenced using the **arrow(->) operator** instead of the **dot(.) operator.**
- When an object pointer is incremented , it points to the next object. When an object pointer is decremented, it points to the previous object.

Using pointer to objects Example 1

```
class student{
    char *name;
    int id;
public:
    void setValues();
    void showValues();
};
void student::setValues(){
    name = (char *)malloc(10*sizeof(char));
    cin>>name;
    cin>>id;
}
void student::showValues(){
    cout<<"Name is: "<<name<<endl;
    cout<<"ID is: "<<id<<endl;
}
```

```
int main(){
        student ob1     ;

        ob1.setValues();


        ob1.showValues();

}
```

Using pointer to objects Example 1

```
class student{
    char *name;
    int id;
public:
    void setValues();
    void showValues();
};
void student::setValues(){
    name = (char *)malloc(10*sizeof(char));
    cin>>name;
    cin>>id;
}
void student::showValues(){
    cout<<"Name is: "<<name<<endl;
    cout<<"ID is: "<<id<<endl;
}
```

```
int main(){
        student ob1;
        student *p;

        p=&ob1;
        p->setValues();
        p->showValues();

}
```

Using pointer to objects Example 2

```
class student{
    char *name;
    int id;
public:
    void setValues();
    void showValues();
};
void student::setValues(){
    name = (char *)malloc(10*sizeof(char));
    cin>>name;
    cin>>id;
}
void student::showValues(){
    cout<<"Name is: "<<name<<endl;
    cout<<"ID is: "<<id<<endl;
}
```

```
int main(){
        student ob[2];

        for(int i=0; i<2;i++){
            ob[i].setValues();
        }
        for(int i=0; i<2;i++){
            ob[i].showValues();
        }
}
```

Using pointer to objects Example 2

```
class student{
    char *name;
    int id;
public:
    void setValues();
    void showValues();
};
void student::setValues(){
    name = (char *)malloc(10*sizeof(char));
    cin>>name;
    cin>>id;
}
void student::showValues(){
    cout<<"Name is: "<<name<<endl;
    cout<<"ID is: "<<id<<endl;
}
```

OUTPUT???

```
int main(){
        student ob[2];
        student *p;

        p=&ob[0];

        for(int i=0; i<2 ; i++){
            p->setValues();
            p++;
        }
        for(int i=0; i<2 ; i++){
            p->showValues();
            p++;
        }

}
```

Using pointer to objects Example 2

F:\programming\Untitled1.exe

```
rahim
21
karim
33
Name is:
Process returned 0 (0x0)   execution time : 9.046 s
Press any key to continue.
```

Output of example 2

Solution??

Using pointer to objects Example 2

```
class student{
    char *name;
    int id;
public:
    void setValues();
    void showValues();
};
void student::setValues(){
    name = (char *)malloc(10*sizeof(char));
    cin>>name;
    cin>>id;
}
void student::showValues(){
    cout<<"Name is: "<<name<<endl;
    cout<<"ID is: "<<id<<endl;
}
```

```
int main(){
        student ob[2];
        student *p;

        p=&ob[0];

        for(int i=0; i<2 ; i++){
            p->setValues();
            p++;
        }
        p=&ob[0];
        for(int i=0; i<2 ; i++){
            p->showValues();
            p++;
        }

}
```

The "**this**" Pointer

- "**this**" pointer is a special type of pointer.
- Automatically passed to any member function when it is called. It is a pointer to the object that generates the call
- The **this pointer** is an implicit parameter to all member functions.
- Only member functions are passed "this" pointer.
- Friend functions are not passed "this" pointer

Example

Ob.f1();

•Ob is the object that calls the function

•The function f1() is passed a pointer to "ob"

Using “this” pointer to objects Example 1

```
class student{
    char *name;
    int id;
public:
    void setValues();
    void showValues();
};
void student::setValues(){
    this->name = (char *)malloc(10*sizeof(char));
    cin>>this->name;
    cin>>this->id;
}
void student::showValues(){
    cout<<"Name is: "<<this->name<<endl;
    cout<<"ID is: "<<this->id<<endl;
}
```

```
int main(){
        student ob1     ;

        ob1.setValues();


        ob1.showValues();

}
```

Passing Objects to a function Example

- Design a Point class which will contain the variables X, Y(private)
- Create a function "SetXY(int a, int b)" to set the value of X and Y.
- Create an object p1 of the class. Set the values.
- Create a function "Distance (Point p2)" which will calculate and show the distance between two points p1 and p2.
- Formula to calculate distance: d = $\sqrt{(X_2 - X)^2 + (Y_2 - Y)^2}$
- Include the "math.h" to use sqrt() function

```
class Point{
private:
    int X;
    int Y;
public:
    void setXY(int a, int b);
    void Distance(Point p2);
};
int main(){
    Point p1,p2;
    p1.setXY(0,0);
    p2.setXY(4,0);
    p1.Distance(p2);
}
```

Solution of Passing Objects to a function Example

```
class Point{
private:
    int X;
    int Y;
public:
    void setXY(int a, int b);
    void Distance(Point p2);
};
void Point::setXY(int a, int b){
    X=a;
    Y=b;
}
void Point::Distance(Point p2){
float dist = sqrt(((p2.X-X)*(p2.X-X)+(p2.Y-Y)*(p2.Y-Y)));
cout<<"Distance between the two points is: "<<dist;
}
```

Does the “p2.X “ and “X” make any confusion??

“this” pointer can be a solution

```
class Point{
private:
    int X;
    int Y;
public:
    void setXY(int a, int b);
    void Distance(Point p2);
};
int main(){
    Point p1,p2;
    p1.setXY(0,0);
    p2.setXY(4,0);
    p1.Distance(p2);
}
```

Solution of Passing Objects to a function Example

```
class Point{
private:
    int X;
    int Y;
public:
    void setXY(int a, int b);
    void Distance(Point p2);
};
void Point::setXY(int a, int b){
    X=a;
    Y=b;
}
void Point::Distance(Point p2){
float dist;
dist=sqrt(((p2.X-this->X)*(p2.X-this->X)+(p2.Y-this->Y)*(p2.Y-this->Y)));
cout<<"Distance between the two points is: "<<dist;
}
```

References

- Can be passed to a function
- Can be returned by a function
- Independent reference can be created

What is a Reference?

- A **reference** variable is an another name for an already existing variable. Once a **reference** is initialized with a variable, either the variable name or the **reference** name may be used to refer to the variable.
- To understand, what a reference parameter is and how it works, let’s first start with a program that uses a pointer as parameter.

Pointer vs Reference

```
void f(int *n){
*n=100;
cout<<"value of n:"<<*n<<endl;
}
int main(){
int i=0;
cout<<"before sending the addr to func:"<<i;
cout<<endl;
f(&i);
cout<<"after returning from func:"<<i;
}
```

```
void f(int &n){
n=100;
cout<<"value of n:"<<n<<endl;
}

int main(){
int i=0;
cout<<"before calling the func:"<<i;
cout<<endl;
f(i);
cout<<"after returning from func:"<<i;
cout<<endl;
}
```

Advantages of using a Reference?

- References don’t need dereferencing operator to access the value. They can be used like normal variables. ‘&’ operator is needed only at the time of declaration.
- Members of an object reference can be accessed with dot operator (‘.’), unlike pointers where arrow operator (->) is needed to access members.
- When an object is passed to a func as a reference, no copy is made. This is one way to eliminate the troubles associated with the copy of an argument damaging something needed elsewhere in the program

Self Study(Section 4.4, Teach yourself c++)

- Show general forms for “**new**” and “**delete**”.
- What are some advantages of using them instead of **malloc**() and **free**()?

Thank You!