Lecture - 6

Scope

- References as aliases
- Returning a reference from a function
- const (Constant) Objects
- const Member functions

References as aliases

- References can also be used as aliases for other variables within a function

```
int main()
{  int count = 1 ;
       int &cref=count ;
       cref++;
       std::cout<<count ;
}
```

- reference variables must be initialized in declarations and cannot be reassigned

Returning a reference from a function

- Problem – the function variables do not have scope outside the function, and memory is de-allocated (***dangling references***)

- The variable should be declared as ***static***

const (Constant) Objects

- Some objects need not be modifiable
- Keyword ***const*** is used to specify that an object is not modifiable
- Attempt to modify the constant object results in a compilation error

Example

```
class time
{
public:
int hour; int minute;
   int seconds;


time(int i,int j,int k) {
   hour=i;minute=j;
   seconds=k; }
}
```

```
void main()
{
const time
    noon(12,0,0);
noon.minute=22;
    //illegal
}
```

const Member function

- Compilers disallow member function calls for ***const objects*** unless the member functions themselves are declared const

- Function is specified as const by inserting the keyword ***const*** after the parameter's list

Example

```
class time
{
private:
int hour; int minute; int
    seconds;
public:
time(int i,int j,int k) {
    hour=i;minute=j;
    seconds=k; }
void display1() const;
void display2();
}
```

```
void time::display2()
{
   std::cout<<hour<<m
   inute<<seconds;}
void time::display1()
   const
{std::cout<<hour<<mi
   nute<<seconds;}
```

```
void main()
{ const time
    noon(12,0,0);
noon.display1();
noon.display2();
    //ERROR
}
```

const Member function

- Compiler does not allow member functions declared const to modify the object

Example

```
class time
{
private:
int hour; int minute; int
    seconds;
public:
time(int i,int j,int k) {
    hour=i;minute=j;
    seconds=k; }
void change1() const;
void display2();
}
```

```
void time::display2()
{
    std::cout<<hour<<
    minute<<seconds;}
void time::change()
    const
{ hour=10; //ERROR}
```

```
void main()
{ const time
    noon(12,0,0);
noon.change();
noon.display2();
    //ERROR
}
```

Constructors/destructors

- Constructors/destructors cannot be const
- However, purpose of these special functions is to modify object
- Special provision – though a constructor must be a non-const member function, it can still be used to initialize a const object

(contd..)

- Invoking a non-const member function from constructor call as part of initialization is allowed

- The “constness” of a const object is enforced from the time the constructor completes initialization of the object until that object’s destructor is called

Class assignment

- How can you control access to different variables in classes (discuss about private and public access specifier)?

- What will happen if you make constructor as private member?