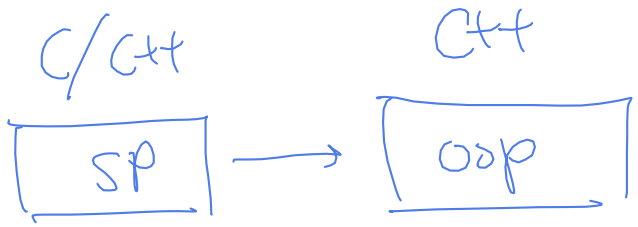


Introduction to Computer & Lab

Lecture No. 6

Function Oriented

VS

Object Oriented

Attributes and Actions

What is an Object? → C++

① driveTo(you, work);

② you.driveTo(work);

'.' 점의 포즈를 함으로써 어떤것이 주체로 어떤행동을 하는지 명확하게 정해주는게 객체지향 프로그래밍임.

An object is

- Something tangible (Ali, Car)
- Something that can be apprehended intellectually (Time, Date)

```
struct Employee {
    short id;
    int age;
    double wage;
};
```

member / field. 메모리할당 X.

Initializing structs

```
Employee joe = {1, 32, 60000.0};
Employee frank = {2, 28};
```

선언한것이 없을땐 0으로 선언됨

Example – Car is a Tangible Object

- State (attributes)
  - Color
  - Model
- behaviour (operations)
  - Accelerate
  - Change Gear
  - Start Car
- Identity
  - Its registration number

포인터는 p[0] → 객체

Arrayname[index] → member

Example – Time is an Object Apprehended Intellectually

- State (attributes)
  - Hours
  - Minutes
  - Seconds
- behaviour (operations)
  - Set Hours
  - Set Minutes
  - Set Seconds
- Identity
  - Would have a unique ID in the model

Data Hiding

Encapsulation

Information Hiding

- Information is stored within the object
- It is hidden from the outside world
- It can only be manipulated by the object itself

```
struct IntArray {
 public:
   int m_array[10];
};

int main() {
   IntArray array;
   array.m_array[16]=2;
```

Invaild array index,
now we over wrote memory that we don't own

```
struct IntArray {
 private:
   int m_array[10];
public:
  void setValue(int index, int value) {
    if (index<0 || index>=10) return;
    m_array[index] = value;
  }
}
```

Example – Information Hiding

사각형 만들 때
upleft (5,9) lowright (-2,4) → 프로그래머의 실수.

컴파일러는 모범이 무엇인지 없으므로 모르게 실행한다.

→ 개발자 입장에서 잘못된 허용을 해서 잘못된 값이 저장되지 않도록 도와야 한다.
또 실수를 했을 때, 실수가 쉽게 발견되도록 해야 한다.

- Ali's name is stored within his brain
- We can't access his name directly
- Rather we can ask him to tell his name

Example – Information Hiding

- A phone stores several phone numbers
- We can’t read the numbers directly from the SIM card
- Rather phone-set reads this information for us

Information Hiding Advantages

- Simplifies the model by hiding implementation details
- It is a barrier against change propagation

Encapsulation

- Data and behaviour are tightly coupled inside an object
- Both the information structure and implementation details of its operations are hidden from the outer world

Example – Encapsulation

- Ali stores his personal information and knows how to translate it to the desired language

- We don’t know
  - How the data is stored
  - How Ali translates this information

Example – Encapsulation

- A Phone stores phone numbers in digital format and knows how to convert it into human-readable characters

- We don’t know
  - How the data is stored
  - How it is converted to human-readable characters

Object has an Interface

struct → Data

- An object encapsulates data and behaviour
- So how objects interact with each other? ↓ using function
- Each object provides an interface (operations)
- Other objects communicate through this interface

Separation of Interface & Implementation

- Means change in implementation does not effect object interface
- This is achieved via principles of information hiding and encapsulation

Example – Separation of Interface & Implementation

- A driver can drive a car independent of engine type (petrol, diesel)
- Because interface does not change with the implementation

Why do we need Constructor

Why do we need Constructor

→The majority of programming problems occur because of the use of un-initialized data.

→ Class is a definition and construct creates object based on class definition.

Constructor

- **Name of the constructor is same as the name of the class**
- **It does not return any thing, not even void**

Example

예약어 public → 클래스의 접근 영역을 표시한다.

클래스에서의 default 접근은 private

프로퍼 에서의 default 접근은 public

```
class Date
{
        int month ;
        int day ;
        int  year ;

  public:
        Date ( int day = 1 , int month = 1 , int year = 1 )
} ;
```

Constructor with default values

Rules of function overloading

When ever we overload a function, the name of the function remain the same but argument list changes.

The argument list can:

- Either vary in the number of arguments
- Or vary in the type

Rules of function overloading applies to constructor overloading

Example

클래스는 정의하는 과정에서 객체의 변수 및 함수의 접근 허용범위를 별도로 선언해야한다. Struct와 class의 차이 "접근 제어 지시자"

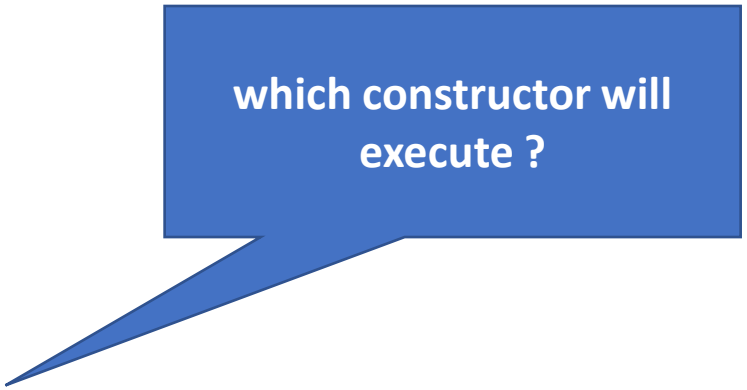
```
class Date
{
  public :
        Date ( int month = 1 , int day = 1 , int year = 2018 ) ;
        Date ( int month, int day=1) ;
        Date (int year) ;
  private :
        int month , day , year ;
} ;
```

public : → 어디서든 접근허용.

private : 클래스 내(클래스 내에 정의된 함수)에서만 접근허용.

Example

```
main ( )
{
  Date mydate ( ) ;
  Date mydate ( 2018);
}
```



Utility Functions

These are functions are used by other function of class

Keep them in private part of class

Example

```
class Date
{
   public :
            Date ( ) ;
            Date ( int month , int day , int year ) ;
            ~Date ( ) ;
            setMonth ( int month ) ;
            setDay ( int day ) ;
            setYear ( int year ) ;
            int getDay ( ) ;
          int getMonth ( ) ;
            int getYear ( ) ;
            setDate (int day, int month, int year ) ;
   private:
            int month , day , year ;
} ;
```

Example

```
main ( )
{
  Date mydate ( 35 , 13 , 2000 ) ;
}
```

As a good programmer always perform validation of values in constructor

Example

```
main ( )
{
  Date mydate (22, 02,2010);
  mydate.setDate ( 21 , 01 , 1979 ) ;
}
```

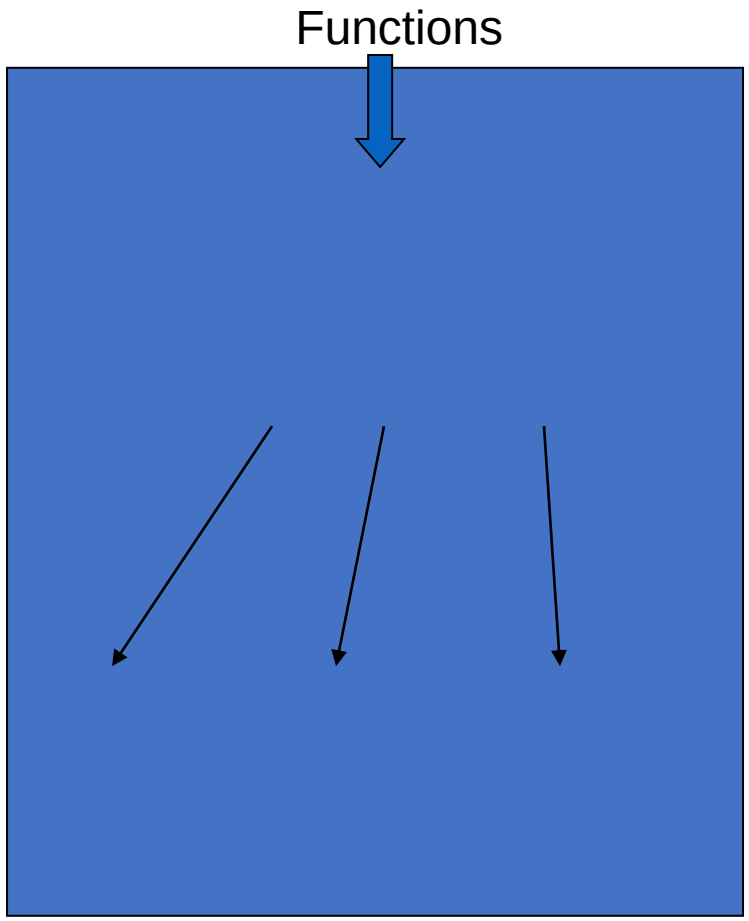
What Happens in Memory

```
setMonth ( int month ) ;
setDay ( int day ) ;
setYear ( int year ) ;
int getMonth ( ) ;
int getDay ( ) ;
int getYear ( ) ;
```

```
int month ;
int day ;
int year ;
```

```
int month ;
int day ;
int year ;
```

```
int month ;
int day ;
int year ;
```



Destructor

~

~className ( ) ;

Rules of Destructor

1. Destructors cannot be overloaded
2. Destructors take no arguments
3. They don’t return a value

Example

```
main ( )
{
  Date date1 , date2 , date3 ;
  date1.setMonth ( 3 ) ;
  date2.setDay ( 23 ) ;
}
```

Destructors

```
~Date :: Date ( )
{
   cout<< “Object Destroyed” ;
}
```

Constructors

```
Date :: Date ( )
{
  cout << “Date Object Created” ;
}
```

Constructors without Arguments

```
Date :: Date ( )
{
  cout<< “Default constructor without
  arguments called ” ;
}
```

Constructors with Arguments

```
Date :: Date ( int month , int day , int year )
{
  cout<< “A constructor with paramerters ” ;
}
```

Memory Allocation

malloc ( ) ;

calloc ( ) ;

realloc ( ) ;

free ( );

```
malloc ( 10 * ( sizeof ( int ) ) ) ;
```

Limitations:

- Initialization is not possible at time of memory allocation
- Returns a void pointer returned
- Casting is required before using

new

new int ;

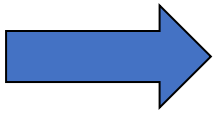
New is an operator not a function

Example

```
int * iptr ;
iptr = new int ;
```

Example

```
int * iptr ;
iptr = new int ;
delete iptr ;
```



Example

```
int *iptr ;
iptr = new int [ 10 ] ;
delete iptr ;
```

Example

```
Date *dptr ;
dptr = new Date ;
```

dptr is a pointer to an object of type date

Example

Will it return the memory consumed by functions and data member?

```
main ( )
{
        Date mydate ;
        cout<< sizeof ( mydate ) ;
}
```

Example

```
int *iptr ;
iptr = new int [ 10 ] ;

Date *dptr ;
dptr = new Date [ 10 ] ;
```

- 10 constructors will be called
- All dynamically created objects will be initialized properly based on constructor's definition.
- It is not possible with calloc/malloc

Example

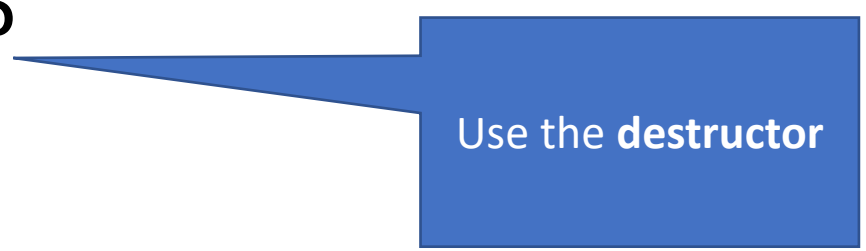
```
Date date1 , *dptr ;
date1.setDate ( ) ;

dptr = new Date ;
dptr ->setDate ( );

dptr.setDate ( ) ;
```

Wrong

Memory Allocation Inside a Constructor

Variable size object

- Use dynamic memory allocation within constructor
- How to release the memory?



char *name =new char [string_length]; //in constructor

delete [ ] name //In destructor

Friend

Friend Function

Friend Function

```
class Date
{
        --

  friend functionName ( Argument_list ) ;
        --
} ;
```

Friendship is granted not acquired

Friend Function

- **Friend functions are NOT the members of the class**
- **The class itself declare it's friend**

  functions
- **The prototype of these functions are**

  written inside the class and the key word friend is appended before the function

  name
- **Friend functions have access to private**

  and public members of the class

Example

```
class myClass
{
  friend increment ( myClass , int ) ;//just increment the
  value of private data
  private:
        int topSecret ;
  public:
        myClass ( ) //constructor
              { topSecret = 100 ; }
        void Display ( )
              { cout<< “The value of topSecret is “
              << topSecret ; }
} ;
```

Example

```
void Increment ( myClass A , int i )
{
  A.topSecret += i ;//can do because it
  is friend
}
```

Example

```
main ( )
{
  myClass x ;
  x.Display ( ) ;
  Increment ( x , 10 ) ;
  x.Display ( ) ;
}
```

Output

The value of topSecret is 100

The value of topSecret is 110

Example

```
class myClassTwo ; //class prototype why we need it?
class myClassOne
{
  private :
            int topSecret ;
  public :
          void Display ( ) ;
          myClassOne ( )
              { topSecret = 100; }
          friend AddBothSecret ( myClassOne , myClassTwo ) ;
} ;
```

Example

```
class myClassTwo
{
  private:
        int topSecret ;
  public:
        void Display ( ) ;
        myClassTwo ( )
              { topSecret = 200 ; }
        friend AddBothSecret ( myClassOne , myClassTwo ) ;
} ;
```

Example

```
main ( )
{
  myClassOne A ;
  myClassTwo B ;
  A.Display ( ) ;
  B.Display ( ) ;
  AddBoth ( A , B ) ;
}
```

Example

```
int AddBoth ( myClassOne A , myClassTwo B )
{
   cout << “The value of topSecret in myClassOne object       is” <<
   A.topSecret ;

   cout << “The value of topSecret in myClassTwo object         is”<<
   B.topSecret ;

   cout << “The sum of topSecret values in “
            << “myClassOne  and myClassTwo object is ”
            << A.topSecret + B.topSecret ;
}
```

Classes can be friends

```
class classTwo;
class classOne
{
  private :
        int topSecret;
  public :
        void Display ( ) ;
        classOne ( )
              { TopSecret = 100 ; }
        friend classTwo ;
} ;
```

Straight Line

$$y = mx + c$$

Straight Line

```
class straightLine
{
    double slope , intercept ;
    // member function
} ;
```

Quadratic

$$y = ax^2 + bx + c$$

Quadratic

```
class quadratic
{
    double a , b , c ;
    // member function
} ;
```

Now a friend of straightLine and quadratic can help to find out line parabola intercept points

Limitations

It is NOT Transitive

It is NOT Associative

Acknowledgment

- Deitel & Deitel :– C++ How to Program, Prentice hall, Inc
- Prof. Naveed A. Malik, Pakistan Institute of Physics