COMS W3101 Programming Language: C++ (Fall 2016)

Ramana Isukapalli
ramana@cs.columbia.edu

Lecture-2

- Overview of C
  - Functions
  - Structures
  - Pointers
- C++
  - Design, difference with C
  - Concepts of Object oriented Programming
  - Concept of class and Object
  - Constructor and destructor
  - Data and Member functions
  - Data encapsulation
    - public, private and protected members

C functions

- A group of statements
  - To perform a task
  - Possibly return a value

- Syntax

```
<return_type> fn_name (arguments)
    {
        // function body
    }
```

C functions … example

- Example

```
int square (int x)  /* fn to compute square*/
{
   return (x * x);
}
void main()         /* Starting point of ANY C program*/
{
   int i = 5;
   int i_sq = square (5);
   cout << "square of 5: " << i_sq << endl;
}
```

C pointers

- A pointer "points" to a memory location.
  - E.g., int x; /* x is an integer */

    int *y; /* y points to an integer */

    x = 5;

    y = 6; / NOT y = 6 ! */
- Pointers can point to any data type;
  - int *x; short *y; char *str;
  - double *z; void *p, etc.

C pointers, contd.

- Why do we need pointers?
  - Mainly to manage the memory, as opposed to the compiler managing memory.
  - User needs to assign and delete memory.
  - Allocate memory using malloc.
  - Delete memory using free.
- Examples
  - int *x = (int *) malloc (sizeof (int));
    *x = 3;
    free (x);

C structs

- C struct
  - used to contain > 1 basic data types
  - Can contain other structs
- E.g.,

```
typedef struct
{
   int a, b, c;
   float x,y,z;
} myStruct;

myStruct m;
m.a = 1;
```

C++

C++ — Philosophically different from C

- High level features of C++
  - Uses concepts of "object oriented programming" (OOP)
  - Everything that works in C works in C++
    - C syntax, operators, structures, control statements, etc. work in C++
    - Reverse is NOT true
- Object Oriented Programming
  - Concept of class/object, methods, inheritance, encapsulation, abstraction, polymorphism
  - Key concepts in this
    - Separation of data and methods

A simple "account" example

```
class account
{
  private:
     int user_SSN;                  // data
     int accountNumber;             // data
  public:
     void withdrawMoney (int amount);   // method
     void depositMoney (int amount);    // method:
     void computeInterest( );           // method
};
account x;  // x is an object of class "account"
```

Constructor and Destructor

Constructor and destructor … contd.

Constructor

- A function with the same name as the class
- Called when an object is created
- A class can have more than one constructor

Destructor

- Called when an object is cleaned up (goes out of scope)
- One class can have only one destructor

Examples

```
account x; // constructor code is called
account *y = new account; // constructor code is called
delete (y); // destructor code is called
```

Constructor and destructor

Constructor code

```
account::account( )
{ user_ssn = -1; accountNumber = -1; }

account::account( ) : user_ssn (-1),
                      accountNumber(-1) { }
account::account (int ssn, int acctNum)
{
    user_ssn = ssn;
    accountNumber = acctNum;
}
```

Destructor code

```
account::~account( )
{  // Any memory/resource cleanup, etc. }
```

Initializing member values

```
class Account
  {
     private:
       int balance;
     public:
       Account ( ) : balance (0) // same as balance = 0
                                 // in the function body { }
         { }
      Account (int amount) : balance (amount) { }
  };
```

Class methods

Syntax:

```
<ret_type> class::functionName(args)
{
   // code
}
```

Methods code can be present eitherin class definition

- In the class defintion or
- Outside the class definition (possibly in a separate file. E.g.,

```
void account::withdrawMoney (int amount)
{
        // code
}
```

Back to "account" example

```
class account
{
  private:
     int user_SSN;            // data
     int accountNumber;       // data
  public:
     account( );                        // Constructor-1
     account(int m, int n);  // Constructor-2
   ~account( );                         // Destructor
     void withdrawMoney (int amount);  // method
     void depositMoney (int amount);    // method:
     void computeInterest( );           // method
};
```

Constructor and destructor

Constructor code

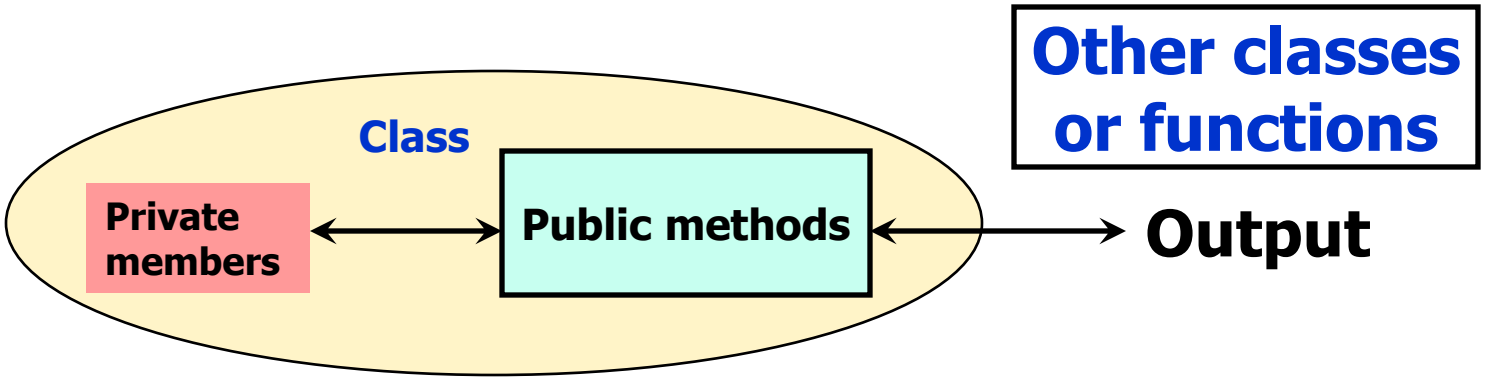
```
    Constructor-1
    account::account( )
    { user_ssn = -1; accountNumber = -1; }
 // OR
    account::account( ) : user_ssn (-1),
                          accountNumber(-1) { }
    // Constructor-2
    account::account (int ssn, int acctNum)
    {
        user_ssn = ssn;
        accountNumber = acctNum;
    }
```

Destructor code

```
    account::~account( )
    {  // Any memory/resource cleanup, etc. }
```

Data encapsulation

Data encapsulation … contd.



- Private members are hidden from other classes, fns.
- Public Methods act on data to provide output.
- External classes, functions have access to public methods
- User should not be affected by
  - Implementation details of public methods.
  - Changes in implementation of methods.

Data encapsulation

- Provide access restrictions to member data and functions
  - From other classes and functions.
- Implemented y using access modifiers
  - **public**, **private** and **protected**
- Other classes, functions need to know what methods are implemented
  - Not how they are implemented

Account example … contd.

- class has both "data" and "methods".
- Attributes and methods are "members" of a class
- An instance of a class is an object.
- A class should typically correspond to some meaningful entity.
- A class uses methods to interact with other classes/functions.
- private members accessible only to the class (and friends)
- public members are accessible to every class and functions

Back to data encapsulation

- How can data be hidden?
  - Only class should have access to data
  - Class methods use data
- Define every class member to be one of
  - public – accessible to every the class, other classes, functions and friends
  - private – accessible only to class and friends
  - protected – accessible only to class, friends and children

Data encapsulation in account example

- In an object of account
  - user_ssn and accountNumber are declared private
    - Accessible only to account objects (and friends)
  - Methods are public
    - Anyone can access them.

- Example

```
void function1 ( ) // function, not defined in Account
 {
        account x;
        x.user_ssn = 123; // Will NOT work
        x.computeInterest ( ); // Will work
}
```