Lecture-4

- C++ string class
- Passing arguments to functions by
  - Value, pointers, refences
- const member functions
- const arguments to a function
- C++ OOP features
  - Data encapsulation
    - public, private and protected members
    - friend functions
    - friend classes
  - Inheritance
- Function overloading and overriding

C++ string class

C++ strings

- C uses char arrays to represent strings
- char arrays are messy
  - Need to predefine the size of array
  - Size can't be increased easily for longer strings.
  - Copying strings need to use strcpy.
- C++ strings – don't have these issues.
  - E.g. string str1 = "abc";

    string str2 = str1;

    string str3 = str1 + "pqr";

  Much more convenient than C character arrays

Passing arguments to a function

Passing arguments to a function

- Passing arguments to function
  - By value – a local copy is made by the function and used.
    - E.g. void function_by value (int arg1)
      - arg1 is passed by value
  - By pointer –address of the argument is used.
    - E.g. void function_by_pointer (int *arg2)
      - Arg2 is passed by pointer – "*" refers to a pointer
  - By reference – similar to passing by pointer
    - E.g. void function_by_reference (int & arg3)
      - Arg3 is passed by reference – "&" refers to reference

Passing arguments… contd.

- void function_by_value (int arg1)
  - Changes made to arg1 inside the function are not seen outside the function.
- void function_by_pointer (int *arg2)
  - Changes made to arg2 inside the function are seen outside the function
- void function_by_reference (int & arg3)
  - Changes made to arg3 inside the function are seen outside the function

const arguments and const member functions

const arguments to functions

```
void f1(const int a)
{
    a = 3; // Not allowed
}
```

- const arguments to a function can't be changed in the function.
- f1 can't change a in the above example

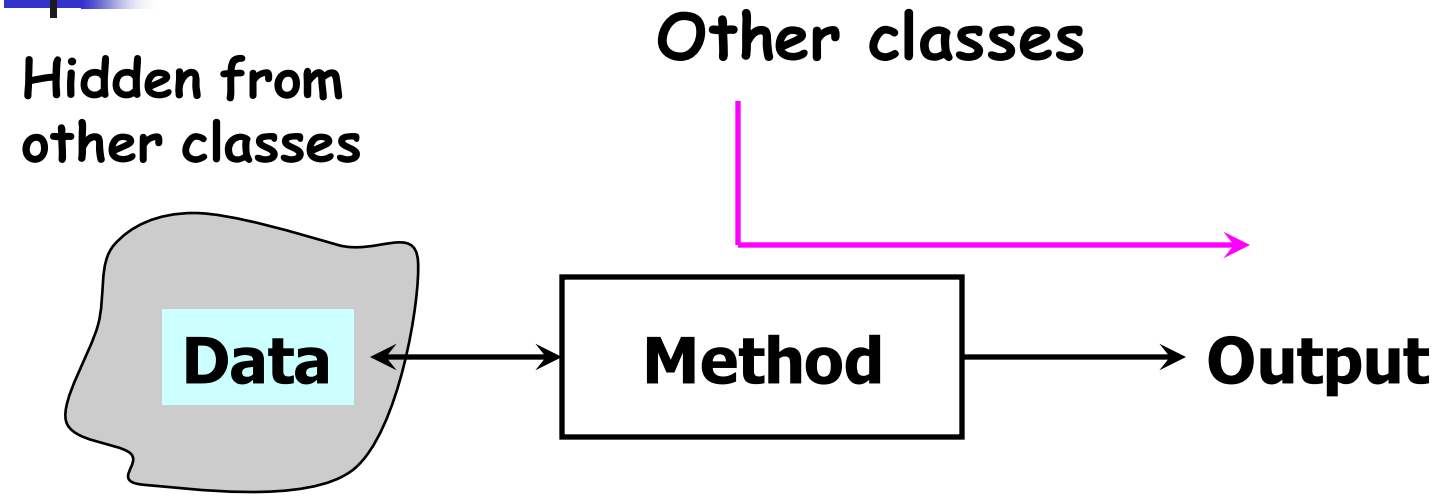
const member functions

```
class myClass
{
   int a;
   …
   void f1( ) const
    { a = 3; } // Not allowed
};
```

- const functions can't change any attributes of myClass.
- f1 can't change a in the above example

Data encapsulation

Data encapsulation … contd.



- Methods act on data to provide output.
- User needs to see only method, not data.
- User should not be affected by
  - Implementation details of 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 class, function
  - 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
}
```

friend functions and friend classes

friend functions

- What if a function genuinely needs to have access to private data?
  - E.g. showAccountInfo (Account acct)
- Need to give access ONLY to that function, not others.
- Use friend function definition
- friend functions of a class have access to private members of the class.

Example – friend function

```
class account
{
private:
    int user_SSN;
    int accountNumber;
public:
    void deposit (int amount)
    void withdraw (int amount);
friend showAccountInfo
    (class Account)
};
```

```
void showAccountInfo
    (Account acct)
{
    cout << user_SSN << endl;
    cout << accountNumber <<
                    endl;
}
```

This is valid.

Friend function can access private members.

friend class

- Concept of friend can be extended to a class from a function.
- A class gives access to its private members to its friend classes.

```
class account
{
  …
   friend class bank
}
```

```
class bank
{
        …
}
```

Members of bank have access to private members of account

Examples

- Valid usage in an external function
  - account acct(123456, 5672);
  - checkingAccount ca;
  - acct.deposit (700);
  - acct.withdraw (300);
  - ca.deposit (1000);
  - ca.showAllChecksCleared( )
- Invalid usage in derived class
  - ca.user_SSN = 1234; // Can't access user_SSN
  - ca.accountNumber = 567;

Inheritance

Inheritance – base class & derived class

- Base class

```
class account
{
    int user_SSN;
    int accountNumber;
public:
    void deposit (int amount);
    void withdraw (int amount);
    double computeInterest ( );
};
```

- Derived class or child class

```
class checkingAccount : public account // checkingAccount is
{                                      // derived from account
    int lastCheckCleared;              // not present in account
    void showAllChecksCleared( );// not present in account
    double computeInterest( ); // defined in both classes
};
```

Inheritance – base class and derived classes

- Base Class

```
class account
{
private:
    int user_SSN;
    int accountNumber;
public:
    account ( ) { }
    account (int ssn, acctNum);
    ~account( ) { }
    void deposit (int amount)
    void withdraw (int amount);
    double computeInterest( );
};
```

- Derived (or child) class-1

```
class checkingAccount : public account
{
public:
    int lastCheckCleared;
    void showChecksCleared ( );
    double computeInterest ( )
};
```

- Derived (or child) class-2

```
class IRA_account : public account
{
public:
    void buyFund (int fund_ID);
    void sellFund (int fund_ID);
    double computeInterest ( );
};
```

Inheritance – continued.

- Important points to note:
  - Derived classes have access to members of base classes in this example.
  - Derived classes can have their own members.
    - E.g. lastCheckCleared, showAllChecksCleared( ), buyFund( ), sellFund( ), etc.
    - Members of one derived class are not accessible to another

Examples

- Valid usage in an external function
  - account acct(123456, 5672);
  - checkingAccount ca;
  - acct.deposit (700);
  - acct.withdraw (300);
  - ca.deposit (1000);
  - ca.showAllChecksCleared( )
- Invalid usage in derived class
  - ca.user_SSN = 1234; // Can't access user_SSN
  - ca.accountNumber = 567;

Function overloading and function overriding

Function overloading

- Functions with the same name but with
  - Different number of arguments or
  - Different types of arguments

E.g. int add (int a, int b, int c) { return (a+b+c); }

int add (int a, int b) { return (a + b); }

double add (double a, double b) { return (a + b); }

- Here "add" is an overloaded function

Function overriding

- Functions defined in parent class and re-implemented by the child class.

E.g.

```
class Bird
{
    int canFly( ) { return (1); }
}
class Penguin
{
    int canFly ( ) { return (0); }
}
```

Here, "canFly" is an overridden by the child class, Penguin