Lecture-3

- Inheritance
- public, private and protected members
- const member functions
- friend functions
- friend classes
- virtual functions
- Polymorphism
- Abstract classes
- Const member functions, const arguments
- Passing arguments by value and reference
- Function overloading

Inheritance

- Let's take the account example again
- There are many types of accounts
 - Checking, saving, money market, IRA, etc.
- All accounts may have
 - Some common members.
 - Account number, user SSN, etc.
 - Some class specific members.
- Method implementation may be
 - Same in different classes
 - Different in different classes.

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
};
                         W3110: Programming Languages – C++
                                Ramana Isukapalli
```

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



- 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;

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:

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.

Members of bank have access to private members of account

virtual functions

- Function "double computeInterest()" is defined in both base and child classes.
 - Supposed to return different values
 - virtual double Account::computeInterest()
 { return 0; }
 - double CheckingAccount::computeInterest() { return 10.0; }
 - double IRA_Account::computeInterest() { return 100.0; }

virtual functions ... contd.

main()

- Account *x = new CheckingAccount(); x→computeInterest(); // Will this return 0 or 10.0?
- This will return
 - 0, if the function is NOT virtual
 - 10.0, if the function is defined virtual

Why are virtual functions needed?

- Mainly to enforce class specific functional implementation.
- Should not call base class function from a child object.
- An account object may take different "forms" at different times
 - Checking account, IRA account, etc.
 - computeInterest() should compute derived class specific function.
- \Rightarrow Polymorphism

Abstract classes

- Consider an object of Account.
- It makes sense to have
 - A specific type (e.g., checking) of account
 - Not just a generic account object.
- A user should be able to create
 - Specific object types.
 - NOT generic objects.
- An abstract class is the generic class.

Abstract classes ... contd.

- Properties of abstract classes.
 - Defines a generic base class
 - Class definition has attributes and methods
 - Other classes are derived from it.
 - Derived classes implement the methods defined in abstract class.
 - Can NOT instantiate objects of base class.
 - Can instantiate only objects of derived classes.

How do we create abstract classes?

```
Set ANY virtual function to 0.
class Account
```

```
virtual double computeInterest () = 0;
}
class CheckingAccount : public Account
{
    double computeInterest ( ) { ... }
}
```

```
Account x; // Will NOT work.
CheckingAccount y; // Will work.
W3110: Programming Languages - C++
Ramana Isukapalli
```

const member functions

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

- const functions can't change any attributes of myClass.
- function_1 can't change a in the above example

Making arguments const

```
class myClass
{
    int a;
    ...
    void f1(const int i )
      { i = 3; } // not allowed
```

Cannot change the value of const arguments

Passing args. to a function ... by value

- Compiler creates its own copy.
- Any changes made inside the function are not reflected after the function. class myClass void f1(int i) // i is passed by value { i = 3; } }; int x = 5; myClass obj; obj.f1(x); cout << "value of x: " << x << endl; // x is still 5 Sep 24, 2008 18 Ramana Isukapall

Passing args. to a function ... by reference

- Compiler takes the original object.
- Any changes made inside the function are reflected after the function.

```
class myClass
{
     void f1(int& i ) // i is passed by reference.
     { i = 3; }
   };
int x = 5;
myClass obj;
obj.f1( x );
cout << "value of x: " << x << endl; // x is 3</pre>
```

Function overloading

 Possible to have multiple member functions of the same name with different parameters
 Function overloading

```
class myClass
{
    // f1 - overloaded function
    void f1 (int i);
    void f1 (int i, int j);
    void f2 (int i, double j);
}
```