CSC230

Intro to C++ Lecture 7

Outline

- Files operation and Lab 4 discussion
- Review: Inheritance
- this pointer
- Overloading and Overriding
- Polymorphism

Files and Streams

- **iostream**
  - cin: standard input
  - cout: standard output

- **fstream**
  - ofstream: represents output file; is used to create files and to write files
  - ifstream: represents input file; is used to read files
  - fstream: represents file stream generally; is a superset of ofstream and ifstream

Open a file

- Either ofstream or fstream can do the job
- When open a file, need to specify the mode of the file

Mode Flag	Description
ios::app	Append mode. Contents will be
	the file.
ios::in	Open the file for reading
ios::out	Open the file for writing
ios::trunc	If the file exists, the contents will be truncated before opening the file

ifstream	read a file that already exist
ofstream	write to a file
fstream	open a file for read/write

These values can be combined by logic OR

Open/close a file

ofstream object

```
ofstream outfile;
outfile.open("file.dat", ios::out | ios::trunc );
```

open() function

filename

Writing mode

"AND"

If file exists, truncate it

```
fstream  afile;
afile.open("file.dat", ios::out | ios::in );
```

```
outfile.close();
afile.close();
```

Closes the file

Read/write a stream

- **Read**: Stream extraction operator (>>)
- **Write**: Stream insertion operator (<<)

```
#include <fstream>
#include <iostream>
using namespace std;

int main ()
{
  char data[100];

  // open a file in write mode.
  ofstream outfile;
  outfile.open("afile.dat");

  cout << "Enter your name: ";
  cin.getline(data, 100);

  // write inputted data into the file.
  outfile << data << endl;

```

```
  cout << "Enter your age: ";
  cin >> data;
  // write inputted data into the file.
  outfile << data << endl;

  // close the opened file.
  outfile.close();
}
```

Read/write a stream

```
#include <fstream>
#include <iostream>
using namespace std;

int main ()
{
  char data[100];
 // open a file in read mode.
  ifstream infile;
  infile.open("afile.dat");

  cout << "Reading from the file" << endl;
  infile >> data;

  // write the data at the screen.
  cout << data << endl;

  //read the data from the file and display it.
  infile >> data;
  cout << data << endl;

  // close the opened file.
  infile.close();
  return 0;
}
```

Example: file-operation.cpp

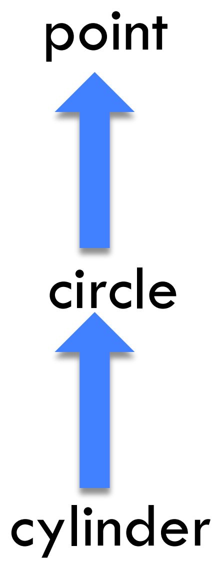
Lab 4 discussion

- Lab 4
  - Create a structure that contains SSN(int), First Name, Last Name;
  - Read data from a file;
  - Store data into a vector;
  - Print out the data;

Outline

- Files operation and Lab 4 discussion
- Review: Inheritance
- this pointer
- Overloading and Overriding
- Polymorphism

Class hierarchy



```
class Point{
  protected:
    int x, y;
  public:
    void set (int a, int b);
};
```

```
class circle : public point{
  private:
    double r;
   ... ...
};
```

```
 class cylinder : public circle{
    private:
      double h;
    ... ...
  };
```

Compare with Java

- Can you inherit from multiple classes in Java?

  class A extend class B, class C ?

- Can you do it in c++?
  - class C: public B, public A
  - **Examples—test_inhertitance2.cpp**

Private member access

```
#include <iostream>
using namespace std;

class father{
  private: int fPrv;
  protected:
    int getPrivateValue(){
      return fPrv;
    }
  };

class son: public father{
  public:
    int foo(){
      return getPrivateValue();
    }
  };

int main(){
  son obj;
  cout<<obj.foo()<<endl;
  cout<<obj.fPrv<<endl;
}
```

Private member in superclass

A protected function access the private member

The protected function is inherited

✓

✗

What is inherited?

- In general, every member of a base class is inherited by a derived class, even the private ones.
  - The private member from base class is not directly accessible to the derived class. It must be through a public/protected method from the base class.

- Some **exceptions**:
  - Constructor and destructor
  - Operator=() member
  - Friends

These functions are class-specific

Rules for constructor/destructor in derived class

Without explicit specification, the default constructor and destructor of the base class will be called first when a new object of the derived class is created or destroyed.

```
class A{
  public:
    A(){
      cout<< "A: default constructor"<<endl;
    }
    A(int a){
      cout<<"A: with a"<<endl;
    }
}
```

```
 class B:public A{
    public:
      B(int a){
        cout<<"B: with a"<<endl;
      }
  }
```

When `B(int a)` is executed, `A()` will be executed first.

If there is a statement in the main():

B obj(1);

The output will be:

A: default constructor
B: with a

Rules for constructor/destructor in derived class

The constructor and destructor of the derived class can specify which constructor/ destructor should be invoked.

```
class A{
  public:
    A(){
      cout<< "A: default constructor"<<endl;
    }
    A(int a){
      cout<<"A: with a"<<endl;
    }
}
```

```
 class B:public A{
    public:
      B(int a) : A(a){
        cout<<"B: with a"<<endl;
      }
  }
```

Example: Test_from_base.cpp

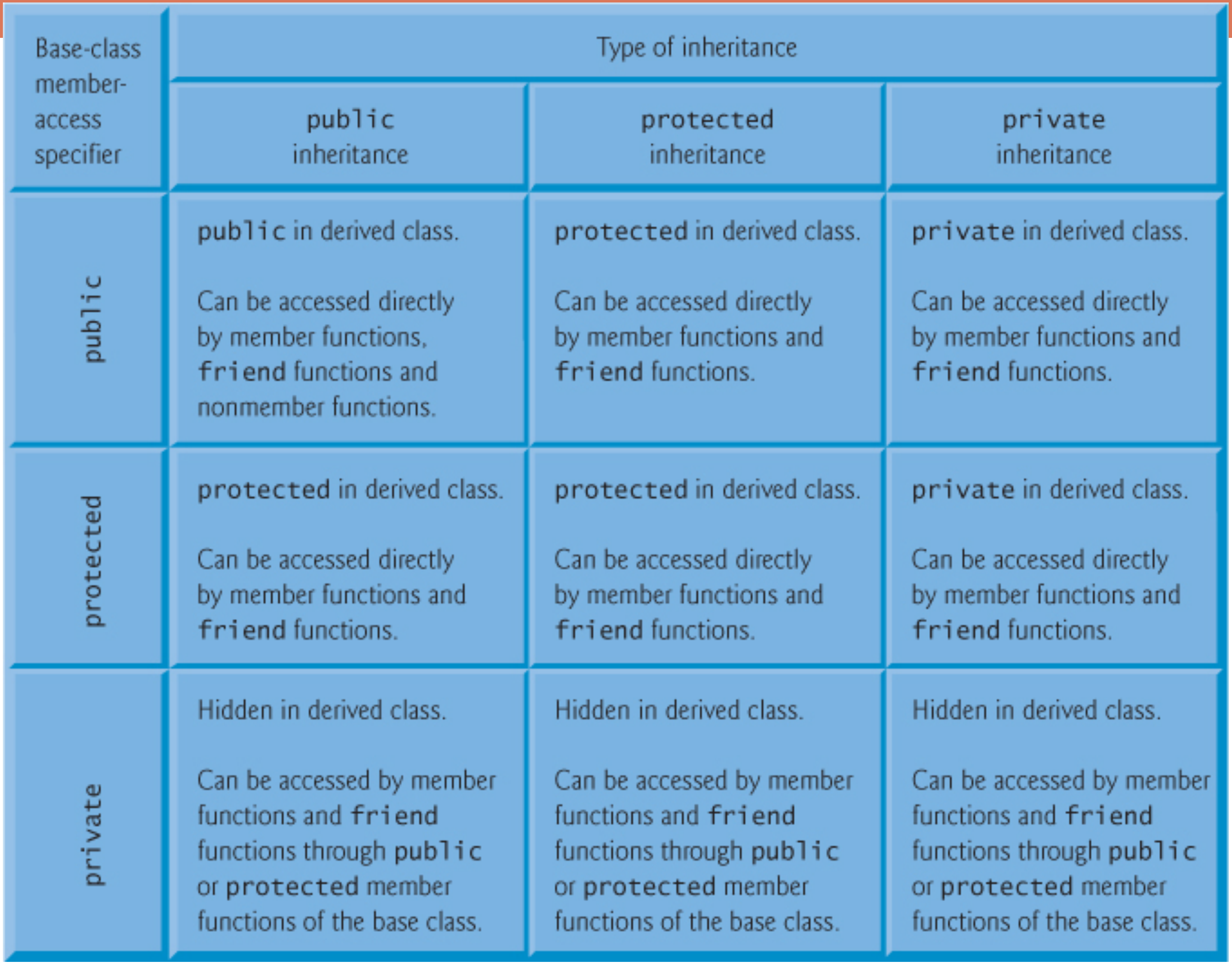
If there is a statement in the main():

B obj(1);

The output will be:

```
A: whit a
B: with a
```

Public & Inheritance



Base-class member-access specifier	Type of inheritance		
	`public` inheritance	`protected` inheritance	`private` inheritance
`public`	`public` in derived class. Can be accessed directly by member functions, `friend` functions and nonmember functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`protected`	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`private`	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.

Outline

- Files operation and Lab 4 discussion
- Review: Inheritance
- this pointer
- Overloading and Overriding
- Polymorphism

this pointer in C++

```
class test{
  private:
    int x;
  public:
    void setX(int x){
      x = x;
    }
};
```

How can we access it from member function?

parameter

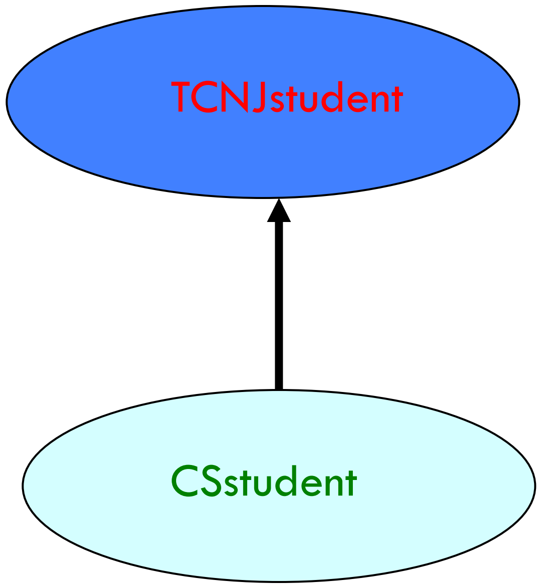
```
class test{
  private:
    int x;
  public:
    void setX(int x){
      this->x = x;
    }
};
```

Example:
Test_this_1.cpp
Test_this_2.cpp

this pointer has the address the current object

Given a object, each member function of this object has a implicit parameter of **this**, only member function has **this**. Friend functions don't have it.

A review of inheritance



Who is the based class?

- TCNJstudent: base class
- CSstudent : the derived class with public inheritance
- A derived class can
  - inherit all members from the base class (correct?)
  - except the constructor
  - access all public and protected members of the base class
  - define its own data member
  - provide its own constructor
  - define its own member functions
  - override functions inherited from the base class

TCNJstudent.h

```
#include <string>
using namespace std;
class TCNJstudent
{
  private:
    string  name;
    string  major;
    int    id;
  public:
    void setName(string a);
    void setMajor(string a);
    TCNJstudent();
    void info();
};
```

TCNJstudent.cpp

```
#include <iostream>
#include <string>
#include "TCNJstudent.h"
using namespace std;

void TCNJstudent::setName(string a)
{ …
}

void TCNJstudent::setMajor(string a)
{
  name = a;
}

TCNJstudent::TCNJstudent()
{ …
}
```

```
void TCNJstudent::info()
{ …
}
```

CSstudent.h

```
#include "TCNJstudent.h"
#include <string>
using namespace std;

class CSstudent : public TCNJstudent
{
 private:
   bool likeGame;
 public:
   CSstudent();
   CSstudent(string a, string b, string c);
   void setMajor();
};
```

Where is the Parent class declaration?

CSstudent.cpp

```
#include <iostream>
#include "CSstudent.h"

void CSstudent::setMajor(){
    TCNJstudent::setMajor("CS");
}

CSstudent::CSstudent(){
  cout <<"From CSstudent()"<<endl;
}

CSstudent::CSstudent(string a, string b, string c){
  cout<< "From CSstudent(a, b, c)" << endl;
}
```

main.cpp

```
#include "CSstudent.h"

int main(){
  CSstudent stu;
  stu.setMajor();
  CSstudent stu2("Mike", "CS", "NJ");
}
```

How to compile them?

Compile

g++ -o excuFile *main.cpp* *TCNJstudent.cpp* **CSstudent.cpp**

Outline

- Files operation and Lab 4 discussion
- Review: Inheritance
- this pointer
- Overload and Overriding
- Polymorphism

Overloading

```
void PrintMe (string s) {
  cout << "string s = \"" << s << "\"" << endl ;
}

void PrintMe (int i) {
  cout << "int i = " << i << endl ;
}
```

Compiler use the signature to decide
which function to use

Overriding

```
class A {
  protected:
    int x, y;
  public:
    void print ()
    {cout<<"From A"<<endl;}
};
```

Method in parent class

Example: overriding1.cpp

```
class B : public A {
  public:
    void print ()
    {cout<<"From B"<<endl;}
};
```

Method in child class with same signature

Overriding

```
class A {
  protected:
    int x, y;
  public:
    void print ()
    {cout<<"From A"<<endl;}
};
```

If we call B b.print(), what will be the output?

```
class B : public A {
  public:
    void print ()
    {
       A::print();
       cout<<"From B"<<endl;
    }
};
```

Another example: overiding2.cpp

Call the print() in A class
NO **super** keyword in C++