CSC230

Intro to C++ Lecture 6

Outline

- Lab 3 / Project 1 discussion
- Object initialization
- Inheritance

Project 1 discussion

Based on Lab 2

- Now, you know how to search a word in the matrix
  - The word should be consecutive in column index
  - With the exception that you can search the next column

Project 1

- In the matrix, each element has its own neighbors
  - left, right, up, down neighbors
  - diagnose neighbors
  - How many neighbors in the candidate set?
  - For the given index number, what are the neighboring elements

Midterm Exam

We will have midterm exam on March 21st.

1. T or F
2. What is the output?
3. Debugging
4. Draw a memory figure.
5. Recursive

Outline

- Lab 3 / Project 1 discussion
- Object initialization
- Inheritance

Static vs. Non-static

non-static data member

Each object has its own copy

static data member

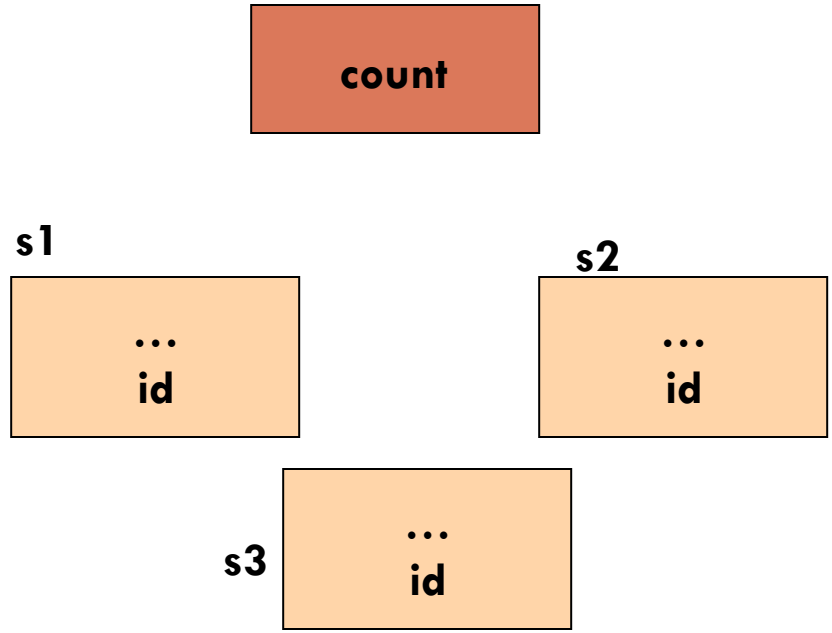
One copy per class type, e.g. counter

```
employee s1;
employee s2;
employee s3;
```

```
class employee
{
  public:
      ...
    int   id;
    static int counter;

    int getID(){
      return id;
    }
};
```

- Examples – static_keyword



Object initialization


```
#include <iostream>

class circle
{
   private:
     double radius;

   public:
     void set(double r);
};
// member function definitions

void circle::set(double r)
{
  radius = r;
}
```

```
int main(void) {
  circle *d;
  d = new circle();
  d->set(5.0);

  circle c;
  c.set(4.0);
}
```

Object initialization, Constructor


```
class circle
{
   private:
     double radius;

   public:
     void set(double r);
     circle();
     circle(const circle &r);
     circle(double r);
};
```

- Default constructor
- Copy constructor
- Constructor with parameters

- **Publicly accessible**
- **same name as the class**
- **no return type**
- **to initialize class data members**
- **different signatures**

Object initialization, Constructor


```
class circle
{
   private:
     double radius;

   public:
     void set(double r);
};
```

When a class is declared with **no constructors**,
the compiler **automatically** assumes default constructor and **copy constructor** for it.

- **Default constructor**

```
circle:: circle() { };
```

- **Copy constructor**

```
circle:: circle (const circle & r)
{
  radius = r.radius;
};
```

Object initialization, Constructor


```
class circle
{
   private:
     double radius;

   public:
     void set(double r);
};
```

- Initialize with default constructor

```
circle r1;
circle *r2 = new circle();
```

- Initialize with copy constructor

•Copy constructor is called when a new object is created from an existing object

•Assignment operator is called when an already initialized object is assigned a new value from another existing object.

If no customer defined constructors. C++ provides default constructors and copy constructor.

Let's check the example, test_copy.cpp

```
circle r3;          //default
r3.set(5.0);

circle r4 = r3;     //copy
circle r5(r4);      //copy

circle *r6 = new circle(r4); //copy
```

Object initialization, Constructor


```
class circle
{
   public:
     double radius;

   public:
     void set(double r);

     circle(double r){radius = r;}

};
```

If **any** constructor is declared,

- no default constructor will exist, unless you define it.
- still have copy constructor

```
circle r1;
```

✗

- Initialize with constructor

```
circle r1(5.0);
circle *r2 = new circle(6.0);
```

✔

Constructor and destructor

An object can be initialized by

- Default constructor
- Copy constructor
- Constructor with parameters

When the object is initialized, resources are allocated.

Just before the object is terminated, the allocated resources should be returned to system.

Destructor


```
class account
{
  private:
    char *name;
    double balance;
    unsigned int id;

  public:
    account();
    account(const account &c);
    account(const char *d);
    ~account();
}

account::~account()
{
  delete[] name;
}
```

destructor:

- Its name is class name preceded by ~
- No argument
- Release dynamic memory and cleanup
- Automatically executed before object goes out of scope, or when delete a pointer to a object.

Example:
test_destructor.cpp
test_con_de.cpp

Destructor declaration

Destructor definition

Delete whole string.

```
delete name;
```

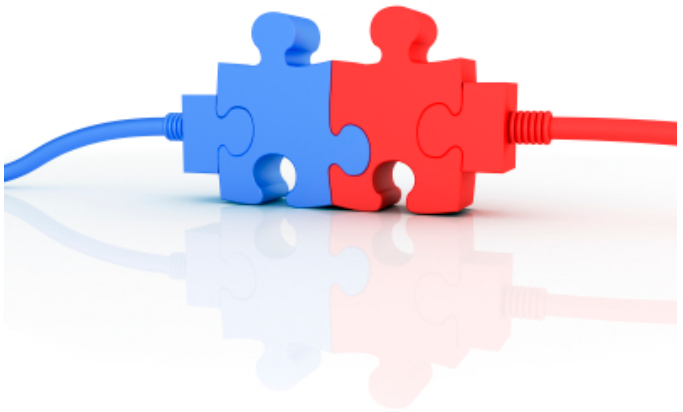
Delete one char.

Work with multiple files

A set of .cpp and .h files for each class group

- .h file contains the prototype of the class
- .cpp contains the implementation of the class

A .cpp file containing the main() function should include all the corresponding .h files where the functions used in .cpp file are declared.



Example: TCNJstudent.h


```
class TCNJstudent
{
  private:
    char  name[50];
    char  major[50];
    int   id;
  public:
    void setName();
    void setMajor();
    TCNJstudent();
    void info() const;
};
```

Example: TCNJstudent.cpp


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

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

Assume the implementation needs this file

Must include the corresponding header file

To simplify the example, we use blank body. A real implementation can have various body.

Example: main.cpp


```
#include "TCNJstudent.h"

int main(){
    …
}
```

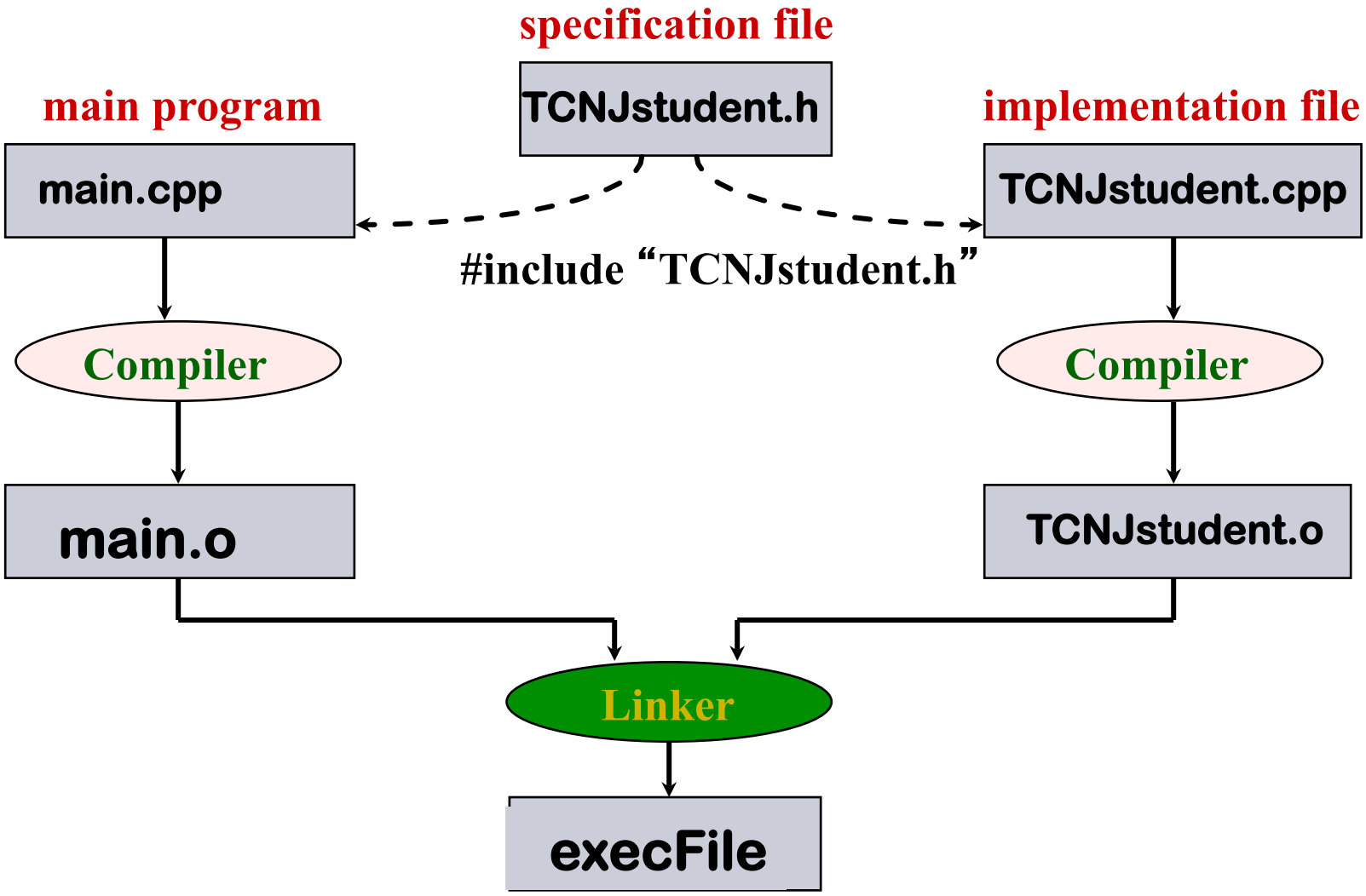
Must include the corresponding header file

Compile

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

Any executable filename you prefer

Separate Compilation and Linking of Files



Review: Parameter passing

Methods for passing parameters

- Passing by values
- Passing by references
- Passing by pointers

Outline

- Lab 3 / Project 1 discussion
- Object initialization
- Inheritance

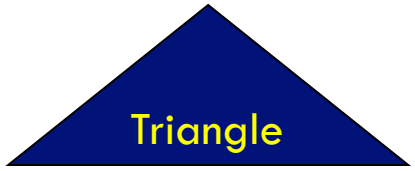
Inheritance

Polygon

```
class Polygon{
  private:
    int numVertices;
    float *xCoord, *yCoord;
  public:
    void set(float *x, float *y, int nV);
};
```

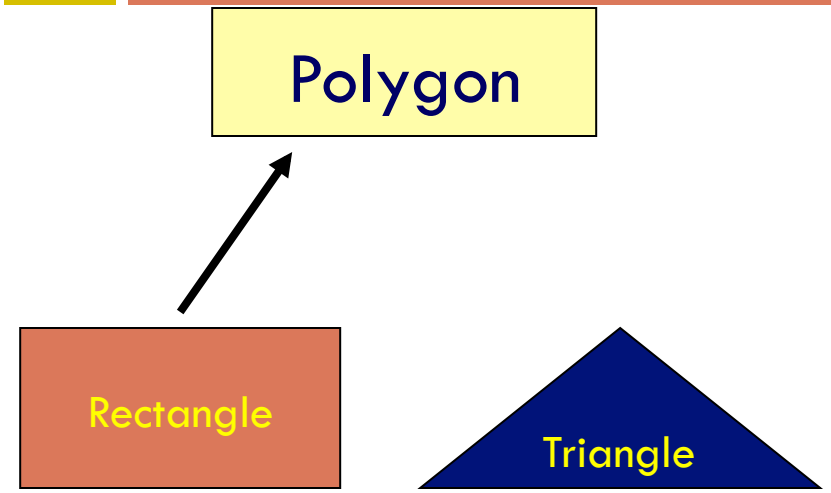
Rectangle

```
class Rectangle{
  private:
    int numVertices;
    float *xCoord, *yCoord;
  public:
    void set(float *x, float *y, int nV);
    float area();
};
```




```
class Triangle{
  private:
    int numVertices;
    float *xCoord, *yCoord;
  public:
    void set(float *x, float *y, int nV);
    float area();
};
```

Inheritance

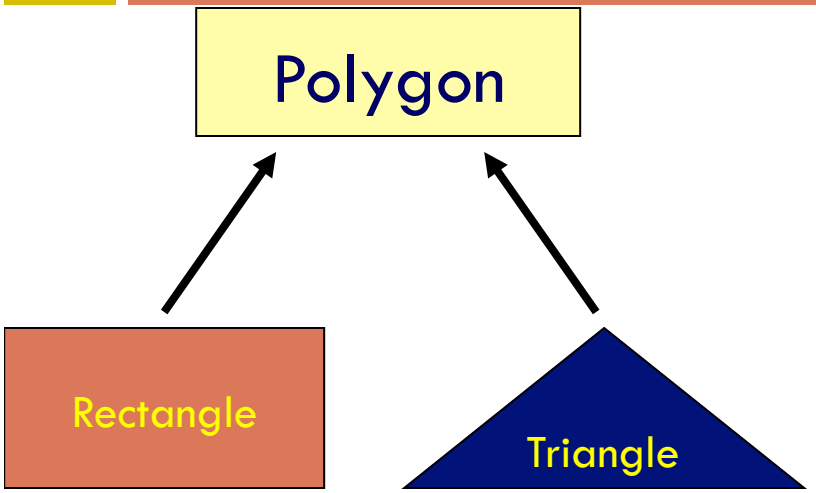



```
class Polygon{
  protected:
    int numVertices;
    float *xCoord, *yCoord;
  public:
    void set(float *x, float *y, int nV);
};
```

```
class Rectangle : public Polygon{
  public:
    float area();
};
```

```
class Rectangle{
  protected:
    int numVertices;
    float *xCoord, *yCoord;
  public:
    void set(float *x, float *y, int nV);
    float area();
};
```

Inheritance




```
class Polygon{
  protected:
    int numVertices;
    float *xCoord, *yCoord;
  public:
    void set(float *x, float *y, int nV);
};
```

```
class Triangle : public Polygon{
  public:
    float area();
};
```

```
class Triangle{
  protected:
    int numVertices;
    float *xCoord, *yCoord;
  public:
    void set(float *x, float *y, int nV);
    float area();
};
```

Base & Derived Classes

- Syntax:

  class *derived-class* : access-specifier *base-class*

  where

  - access-specifier is one of **public, protected**, or **private**
    - private by default
    - Most of the time, people use public
- Any class can serve as a base class
  - Thus a derived class can also be a base class

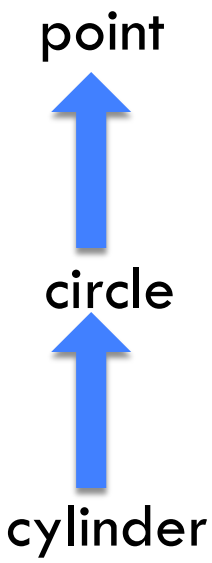
Friend keyword

- Friend
  - In principle, private and protected members of a class cannot be accessed from outside the same class in which they are declared. However, this rule does not apply to "friends".
  - Friends are functions or classes declared with the friend keyword.
  - Example: test_friend_1/2.cpp

Public & Inheritance

When the component is declared as:	When the class is inherited as:	The resulting access inside the subclass is:
public	public	Public
protected		protected
private		none
public	protected	protected
protected		protected
private		none
public	private	private
protected		private
private		none

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?

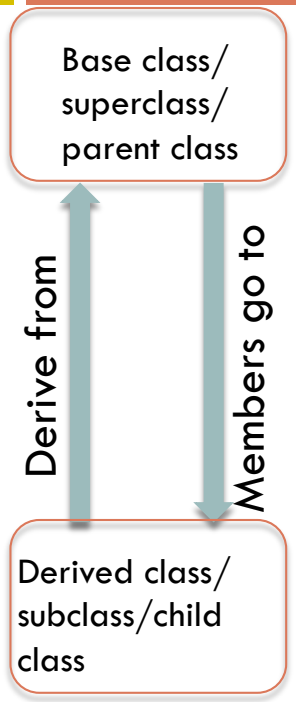
  class A extend class B, class C ?

- Can you do it in c++?
  - class C: public B, public A
  - **Examples—test_inhertitance1/2.cpp**

What to inherit?

- In principle, every member of a base class is inherited by a derived class
  - just with different access permission

Access control over the members



How to decide the **access specifiers** of the **members** in the **subclass**?

- Class definition
- Inheritance type

```
class Polygon{
  protected:
    int numVertices;
    float *xCoord, *yCoord;
  public:
    void set(float *x, float *y, int nV);
};
```

```
class Triangle : public Polygon{
  public:
    float area();
};
```

Access specifier of derived class

class *derived-class* : access-specifier *base-class*

Type of Inheritance

Access specifier of superclass member

	Public	**Protected**	**Private**
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	-	-	-

- The type of inheritance defines the access level for the members of derived class that are inherited from the base class

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

Midterm Exam

We will have midterm exam on March 21st.

1. T or F
2. What is the output?
3. Debugging
4. Draw a memory figure.
5. Recursive