CSC230

Intro to C++ Lecture 4

Outline

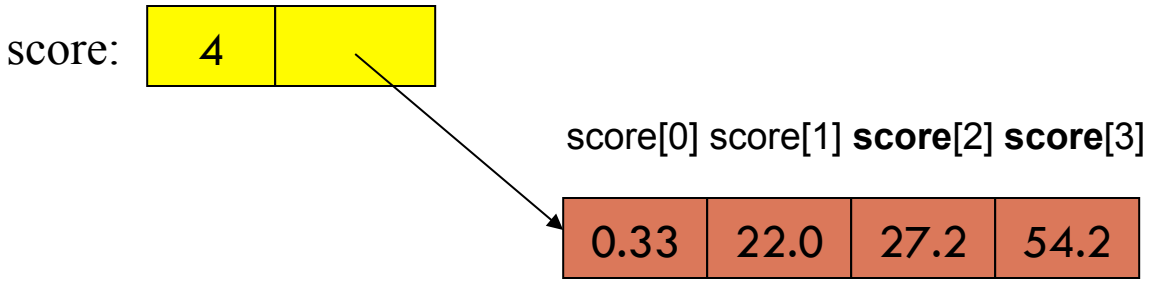
- Lab 2 / Project 1 discussion
- Structure
- Class and Object
- Static keyword

Vector

- A **vector**
  - Can hold an **arbitrary** number of elements
    - Up to whatever physical memory and the operating system can handle
  - That number can **vary** over time
    - E.g. by using **push_back()**
  - Example

    vector<double> score(4);

    score[0]=.33; score[1]=22.0; score[2]=27.2; score[3]=54.2;



Array vs. vector

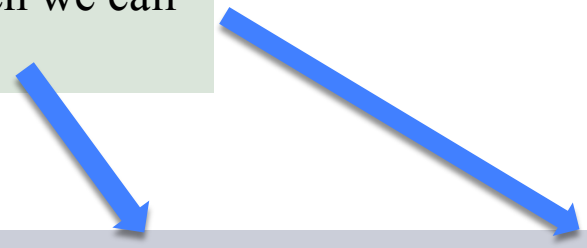
Array	Vector
Provides contiguous, indexable sequence of elements	Provides contiguous, indexable sequence of elements
Once created, the size cannot be changed	Size change be changed, grow or shrink dynamically
If dynamically allocated, user got a pointer, the user can use *sizeof(arr)/ sizeof(*arr)* to figure out the array size. But it is error-prone.	When a vector is created, one object is created. A vector object is not a pointer, but *&vec[0]*returns the starting address of the data
If the array is dynamically allocated, user need to de-allocate it.	Vector automatically manages memory, including allocation and de-allocation.
Usually when passed to a function, it is passed as a pointer with separate parameters for its size. Cannot be returned from a function.	Can be passed to/returned from function
Can't be copied/assigned directly	Can be copied/assigned directly

Revisit a 2D array parameter example

address of element (r, c) = base address of array
+ r*(number of elements in a row)*(size of an element)
+ c*(size of an element)

What if we do not know the col value?

- col value must be determined when we define the function
- row value can be passed when we call the function



```
void init(int twoD[][col], const int row) {
  for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++)
      twoD[i][j] = -1;
  }
}
```

Pass a 2D vector as parameter


```
void init(vector< vector<char> > &twoD) {
  for (int i = 0; i < twoD.size(); i++) {
    for (int j = 0; j < twoD[0].size(); j++)
      twoD[i][j] = -1;
  }
}
```

twoD is a reference to 2D vector of char

twoD element is accessed like 2D array

```
vector< vector<char> > searchMatrix;
searchMatrix.resize(x);
for (int i=0; i<x; i++) {
    searchMatrix[i].resize(y);
}
....
init(searchMatrix);
```

Declare 2D vector

First dimension size is x

The element of the first dimension is a vector with size y

Lab 2/Project 1 discussion: Vector

Vector of a vector

- 2D Vector Declaration : vector < vector < char> > Test_1

Test_1__Row_1
Test_1__Row_2
Test_1__Row_3
Test_1__Row_4

- Example -- test_vector.cpp

Each row (the inner vector) is independent with each other

Test_1__Row_1
Test_1__Row_2
Test_1__Row_3
Test_1__Row_4

Lab 2/Project 1 discussion: Vector

Vector of a vector

- 2D Vector initialization:

```
int m, n;

cin>>m>>n;

vector<vector<int> > v;

for(int i=0; i<m; i++)
{
    for(int j=0; j<n; j++)
    {
        int a;
        cin>>a;
        v[i].push_back(a);
    }
}
```

Access element through index need to be initialize first

```
for (int i=0; i<m; i++)
{
    v.push_back(vector<int>());
    for (int j=0; j<n; j++)
    {
        int a;
        cin >> a;
        v[i].push_back(a);
    }
}
```

```
vector<vector<int> > v(m);

for(int i=0; i<m; i++)
{
    for(int j=0; j<n; j++)
    {
        int a;
        cin>>a;

        v[i].push_back(a);
    }
}
```

Initialize v with m rows

Lab 2/Project 1 discussion: Vector

Vector of a vector

- index vs push_back

index : the element has to be there (initialize it before you use it)
push_back : append a value at the end of the vector

- size of the 2d vector

  vector<vector<char> > Test_1;

- what is the Test_1.size() ?
- what is the Test_1[0].size()?
- Example -- test_vector.cpp

Lab 2/Project 1 discussion: Vector

Traverse the 2D vector

- for loop in 2D vector

```
for(int i=0; i<ROW; i++)
   {
      rowvector.clear();

      for(int j=0; j<COL; j++) {

            cin >> current;

rowvector.push_back(current);

                        }

array2.push_back(rowvector);
   }
```

What is the starting point of the loop?

What if we do not know the total row number or how many elements in each row?

How to change the starting point to x and y?

Lab 2/Project 1: arguments to main

Pass arguments to main function

- main (int argc, char *argv[])
- Examples – test_main.cpp / test_1.cpp
- Project 1 discussion

Outline

- Lab 2 / Project 1 discussion
- Structure
- Class and Object
- Static keyword

Structure

array:

- User defines
- Combine multiple data items of same type

structure:

- User defines
- Combine multiple data items of different types

```
struct TCNJstudent
{
  char  name[50];
  char  major[50];
  char  homeAddress[100];
  int   id;
}csStudent, mathStudent;
```

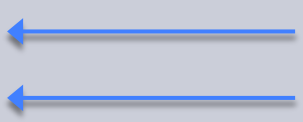
Structure tag (optional)

Member definition

Structure variable(s)

Access members of a structure


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

struct TCNJstudent
{
  char  name[50];
  char  major[50];
  char  homeAddress[100];
  int   id;
};

int main ()
{
  struct TCNJstudent csStudent, mathStudent;
  csStudent.id = 1000;
  mathStudent.id = 2000;
  strcpy(csStudent.name, "Mike Lee");
  strcpy(csStudent.major, "CS");
  strcpy(csStudent.homeAddress, "Earth");
  cout << csStudent.name << " " << csStudent.homeAddress <<endl;
  return 0;
}
```



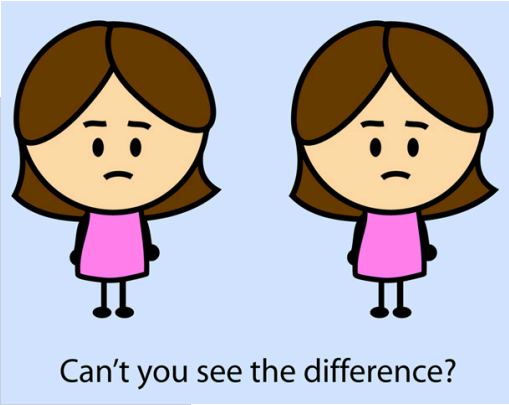
Structure as a function parameter


```
struct TCNJstudent
{
  char  name[50];
  char  major[50];
  char  homeAddress[100];
  int   id;
};

void infoCheck(struct TCNJstudent student)
{
  cout << student.name << endl;
}
```

```
infoCheck(csStudent);
```

Pointers to structures


```
struct TCNJstudent
{
  char  name[50];
  char  major[50];
  char  homeAddress[100];
  int   id;
};

void infoCheck(struct TCNJstudent *student)
{
  cout << student->name << endl;
}
```




```
infoCheck(&csStudent);
```

- Examples – pointer_struct.cpp

Outline

- Lab 2 / Project 1 discussion
- Structure
- Class and Object
- Static keyword

Class and Object


```
class Base {

  public:

  // public members go here

  protected:

  // protected members go here

  private:

  // private members go here

};
```

- Access specifiers: public, private, protected
- Each class may have multiple sections
- Each section remains effective until either another section or the end of the class body
- The default access is private

Class and object example


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

class student
{
  public:
    char  name[50];
    char  major[50];
    char  homeAddress[100];
};

int main ()
{
  student csStudent, mathStudent;
  strcpy(csStudent.name, "Mike Lee");
  strcpy(csStudent.major, "CS");
  strcpy(csStudent.homeAddress, "Earth");
  cout << csStudent.name << " " << csStudent.homeAddress <<endl;
  return 0;
}
```

Method definition


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

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

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

    int getID();
};

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

declaration

definition

scope operator

Const method/member function

- **const** method/member function
  - declaration
    - *return_type func_name* (*para_list*) **const;**
  - definition
    - *return_type func_name* (*para_list*) **const** { … }
    - *return_type class_name* :: *func_name* (*para_list*) **const** { … }
  - It is **illegal** for a const member function to **modify** a class data member

Example: const_keyword.cpp

Const Member Function


```
class  student
{
 private :
  string name, addr, major;
 public :
    void info() const;
};
```

function declaration

function definition

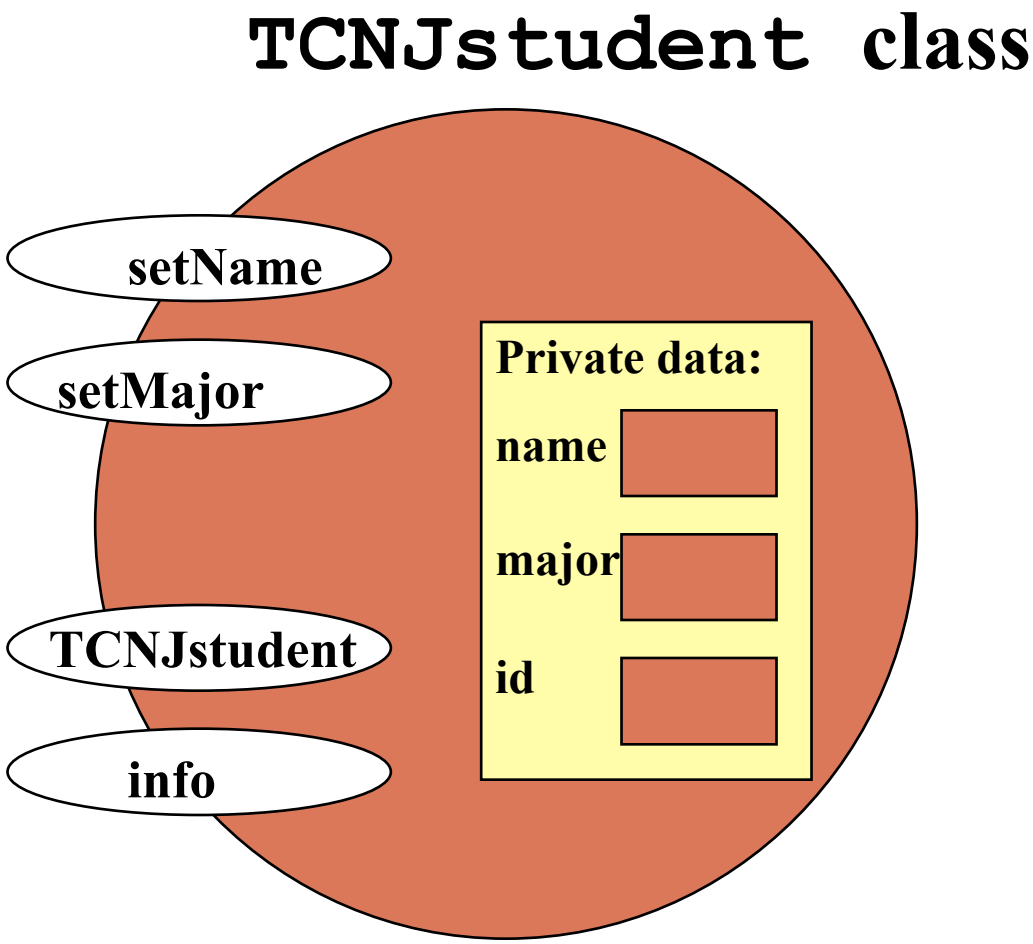
```
void student:: info( ) const
{
  cout << name << ":" << addr << ":" << major << endl;
}
```

TCNJstudent class


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

Bugs?

Class Interface



Access specifier

Access From	Public	Protected	Private
Same class	Yes	Yes	Yes
Derived classes	Yes	Yes	No
Everywhere	Yes	No	No

- The **default** access specifier is private.
- The data members are usually private or protected. A private member can be accessed by another member function of the same class (exception friend function, more details later)
- Each access control section is optional, repeatable, and sections may occur in any order

One more example


```
#include <iostream>
class circle
{
   private:
     double radius;
   public:
     void store(double);
     double area(void);
     void display(void);
};
```

```
// member function definitions
void circle::store(double r)
{
  radius = r;
}

double circle::area(void)
{
  return 3.14*radius*radius;
}

void circle::display(void)
{
  std::cout << "r = " << radius << std::endl;
}
```

```
int main(void) {
  circle c;   // an object of circle class
  c.store(5.0);
  std::cout << "The area of circle c is " << c.area() << std::endl;
  c.display();
}
```

Look inside the example


```
int main(void) {
  circle c;   // an object of circle class
  c.store(5.0);
  std::cout << "The area of circle c is " << c.area() << std::endl;
  c.display();
}
```

c is **statically** allocated

endl is defined in std **namespace**

Does this one work?


```
int main(void) {
  circle c, *d;
  d.store(5.0);
  std::cout << "The area of circle c is " << d.area() << std::endl;
  d.display();
}
```

- d is a pointer, which should have the address of someone in the memory.
- Did we initialize d ? NO!
- Did compile initialize it? NO!

First modification


```
int main(void) {
  circle c, *d;
  d = &c;
  d.store(5.0);
  std::cout << "The area of circle c is " << d.area() << std::endl;
  d.display();
}
```

- d is initialized
- d is a pointer, we **cannot** use d.store(), d.area(), d.display() to access the functions.

Second modification


```
int main(void) {
  circle c, *d;
  d = &c;
  d->store(5.0);
  std::cout << "The area of circle c is " << d->area() << std::endl;
  d->display();
}
```

Outline

- Lab 2 / Project 1 discussion
- Structure
- Class and Object
- Static keyword

Static vs. Non-static

non-static data member

Each object has its own copy

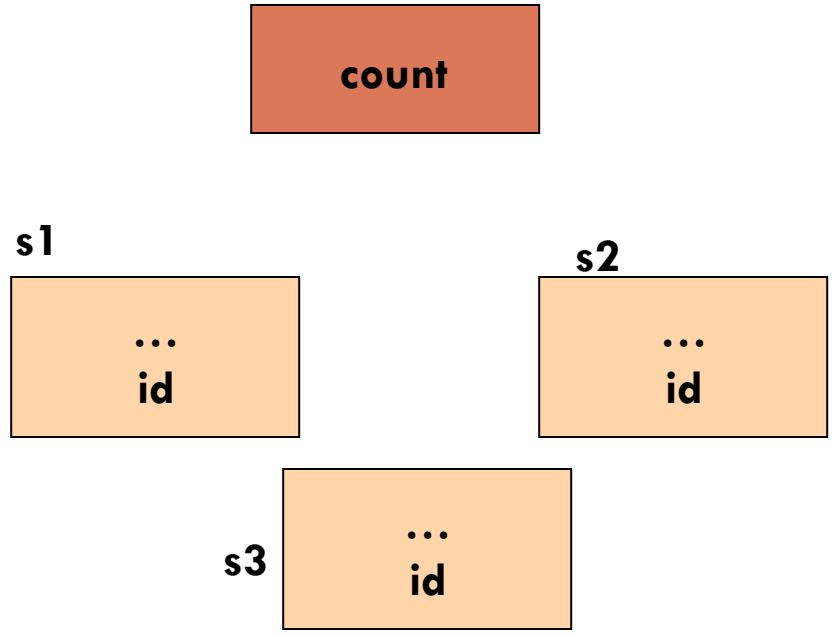
static data member

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;
}
```

```
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.

Destructor declaration

Destructor definition

Delete whole string.

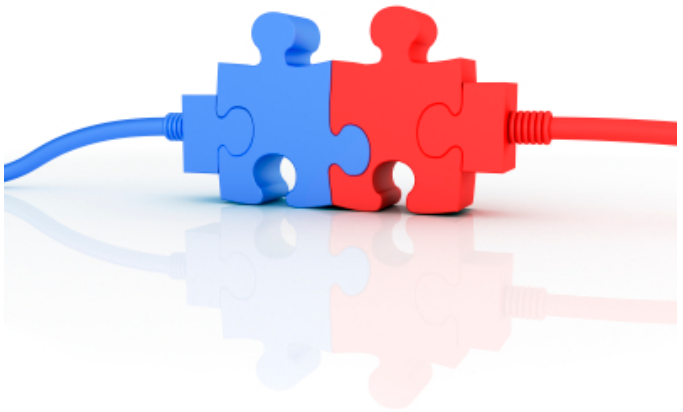
Delete one char.

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