PROGRAMMING WITH C++

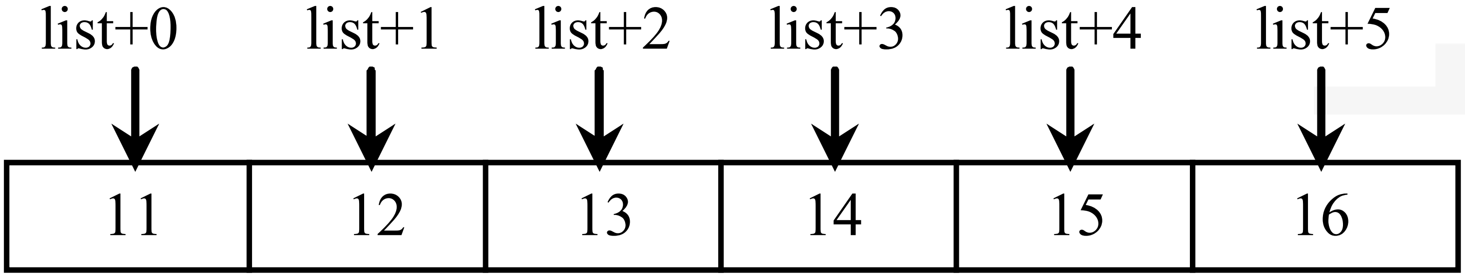
LAB 10

Xi’an Jiaotong-Liverpool University
西交利物浦大学

Example 1: Arrays And Pointers

An array variable without a bracket and a subscript actually represents the starting address of the array. In this sense, an array variable is essentially a pointer. Suppose you declare an array of int value as follows:

int list[6] = {11, 12, 13, 14, 15, 16};



list+0	list+1	list+2	list+3	list+4	list+5
11	12	13	14	15	16

ARRAY POINTER

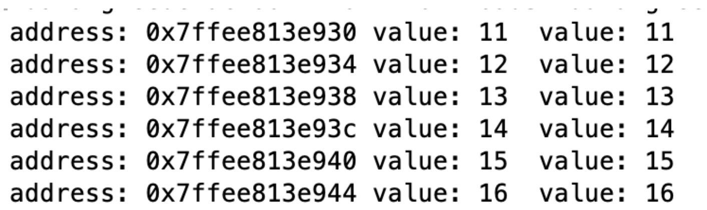
*(list + 1) is different from *list + 1. The dereference operator (*) has precedence over +. So, *list + 1 adds 1 to the value of the first element in the array, while *(list + 1) dereference the element at address (list + 1) in the array.

1. Write a complete program that uses pointers to access array elements.

2. Arrays and pointers form a close relationship. A pointer for an array can be used just like an array. You can even use pointer with index.

ArrayPointer.cpp

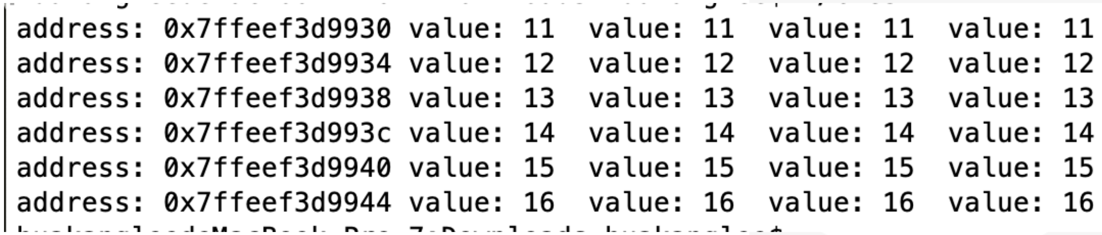
```
#include <iostream>
using namespace std;

int main()
{
  int list[6] = {11, 12, 13, 14, 15, 16};

  for (int i = 0; i < 6; i++)
    cout << "address: " << (list + i) <<
      " value: " << *(list + i) << " " <<
      " value: " << list[i] << endl;

  return 0;
}
```



```
address: 0x7ffee813e930 value: 11  value: 11
address: 0x7ffee813e934 value: 12  value: 12
address: 0x7ffee813e938 value: 13  value: 13
address: 0x7ffee813e93c value: 14  value: 14
address: 0x7ffee813e940 value: 15  value: 15
address: 0x7ffee813e944 value: 16  value: 16
```

PointerWithIndex.cpp

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

int main()
{
  int list[6] = {11, 12, 13, 14, 15, 16};
  int* p = list;

  for (int i = 0; i < 6; i++)
    cout << "address: " << (list + i) <<
      " value: " << *(list + i) << " " <<
      " value: " << list[i] << " " <<
      " value: " << *(p + i) << " " <<
      " value: " << p[i] << endl;

  return 0;
}
```



```
address: 0x7ffeef3d9930 value: 11  value: 11  value: 11  value: 11
address: 0x7ffeef3d9934 value: 12  value: 12  value: 12  value: 12
address: 0x7ffeef3d9938 value: 13  value: 13  value: 13  value: 13
address: 0x7ffeef3d993c value: 14  value: 14  value: 14  value: 14
address: 0x7ffeef3d9940 value: 15  value: 15  value: 15  value: 15
address: 0x7ffeef3d9944 value: 16  value: 16  value: 16  value: 16
```

Example 2: Passing Pointer Arguments

A pointer argument can be passed by value or by reference. For example, you can define a function as follows:

```
void f(int* p1, int* &p2)
```

which is equivalently to

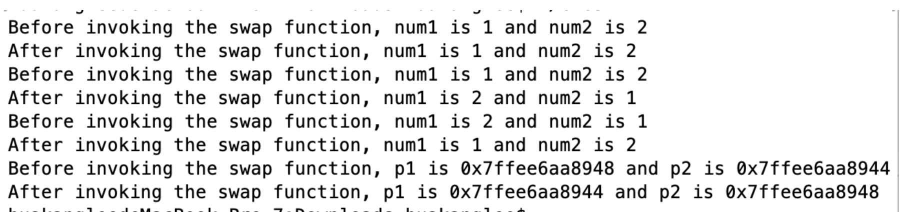
```
typedef int* intPointer;
void f(intPointer p1, intPointer& p2)
```

Here p1 is pass-by-value and p2 is pass-by-reference.

We now give an example of passing pointers in swap function to demonstrate the effect.

TestPointerArgument.cpp

```
#include <iostream>
using namespace std;
// Swap two variables using pass-by-value
void swap1(int n1, int n2){
  int temp = n1;
  n1 = n2;
  n2 = temp;
}
// Swap two variables using pass-by-reference
void swap2(int& n1, int& n2){
  int temp = n1;
  n1 = n2;
  n2 = temp;
}
// Pass two pointers by value
void swap3(int* p1, int* p2){
  int temp = *p1;
  *p1 = *p2;
  *p2 = temp;
}
// Pass two pointers by reference
void swap4(int* &p1, int* &p2){
  int* temp = p1;
  p1 = p2;
  p2 = temp;
}
```



```
Before invoking the swap function, num1 is 1 and num2 is 2
After invoking the swap function, num1 is 1 and num2 is 2
Before invoking the swap function, num1 is 1 and num2 is 2
After invoking the swap function, num1 is 2 and num2 is 1
Before invoking the swap function, num1 is 2 and num2 is 1
After invoking the swap function, num1 is 1 and num2 is 2
Before invoking the swap function, p1 is 0x7ffee6aa8948 and p2 is 0x7ffee6aa8944
After invoking the swap function, p1 is 0x7ffee6aa8944 and p2 is 0x7ffee6aa8948
```

```
int main()
{
  // Declare and initialize variables
  int num1 = 1;
  int num2 = 2;
  cout << "Before invoking the swap function, num1 is "
    << num1 << " and num2 is " << num2 << endl;
  // Invoke the swap function to attempt to swap two variables
  swap1(num1, num2);
  cout << "After invoking the swap function, num1 is " << num1 <<
    " and num2 is " << num2 << endl;
  cout << "Before invoking the swap function, num1 is "
    << num1 << " and num2 is " << num2 << endl;
  // Invoke the swap function to attempt to swap two variables
  swap2(num1, num2);
  cout << "After invoking the swap function, num1 is " << num1 <<
    " and num2 is " << num2 << endl;
  cout << "Before invoking the swap function, num1 is "
    << num1 << " and num2 is " << num2 << endl;
  // Invoke the swap function to attempt to swap two variables
  swap3(&num1, &num2);
  cout << "After invoking the swap function, num1 is " << num1 <<
    " and num2 is " << num2 << endl;
  int* p1 = &num1;
  int* p2 = &num2;
  cout << "Before invoking the swap function, p1 is "
    << p1 << " and p2 is " << p2 << endl;
  // Invoke the swap function to attempt to swap two variables
  swap4(p1, p2);
  cout << "After invoking the swap function, p1 is " << p1 <<
    " and p2 is " << p2 << endl;
  return 0;
}
```

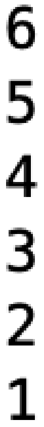
Example 3: Returning A Pointer From Functions

You can use pointers as parameters in a function. Can you return a pointer from a function? The answer is yes.

ReverseArrayUsingPointer.cpp

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

int* reverse(int* list, int size)
{
  for (int i = 0, j = size - 1; i < j; i++, j--)
  {
    // Swap list[i] with list[j]
    int temp = list[j];
    list[j] = list[i];
    list[i] = temp;
  }

  return list;
}

void printArray(const int* list, int size)
{
  for (int i = 0; i < size; i++)
    cout << list[i] << " "<<endl;
}
```

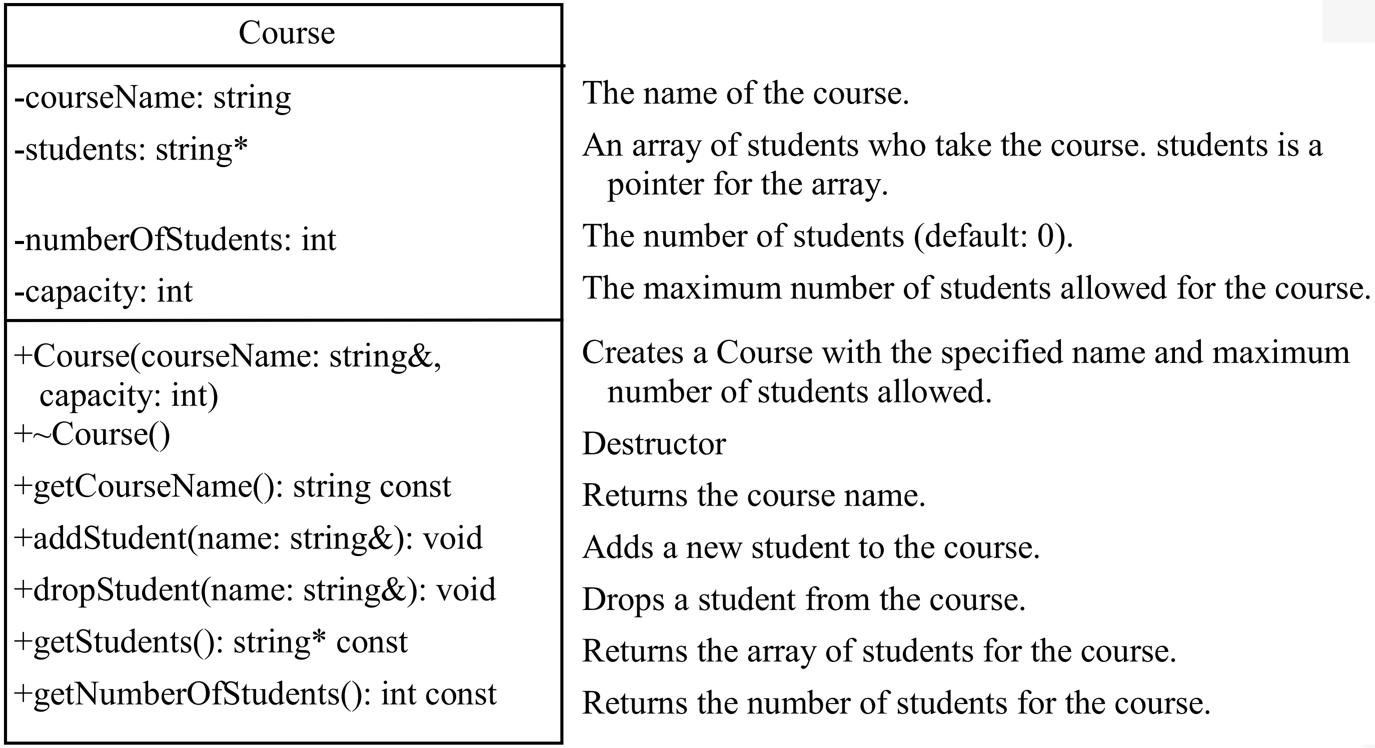
```
int main()
{
  int list[] = {1, 2, 3, 4, 5, 6};
  int* p = reverse(list, 6);
  printArray(p, 6);

  return 0;
}
```



```
6
5
4
3
2
1
```

Example 4: The Course Class with String

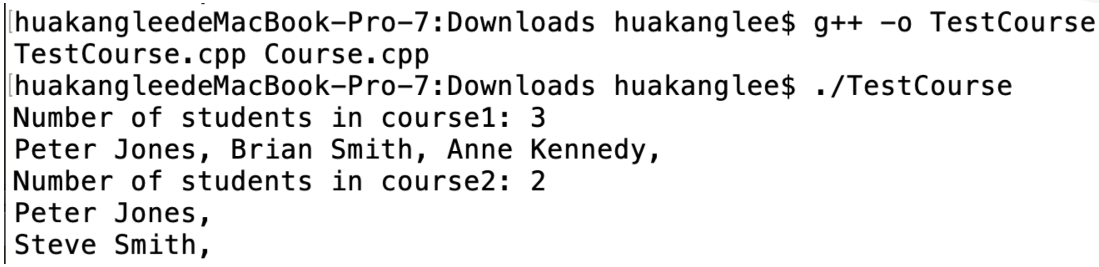
Suppose you need to process course information. Each course has a name and a number of students who take the course. You should be able to add/drop a student to/from the course. You can use a class to model the courses



Course	
-courseName: string	The name of the course.
-students: string*	An array of students who take the course. students is a pointer for the array.
-numberOfStudents: int	The number of students (default: 0).
-capacity: int	The maximum number of students allowed for the course.
+Course(courseName: string&, capacity: int)	Creates a Course with the specified name and maximum number of students allowed.
+~Course()	Destructor
+getCourseName(): string const	Returns the course name.
+addStudent(name: string&): void	Adds a new student to the course.
+dropStudent(name: string&): void	Drops a student from the course.
+getStudents(): string* const	Returns the array of students for the course.
+getNumberOfStudents(): int const	Returns the number of students for the course.

TestCourse.cpp

```cpp
#include <iostream>
#include "Course.h"
using namespace std;
int main()
{
  Course course1("Data Structures", 10);
  Course course2("Database Systems", 15);
  course1.addStudent("Peter Jones");
  course1.addStudent("Brian Smith");
  course1.addStudent("Anne Kennedy");
  course2.addStudent("Peter Jones");
  course2.addStudent("Steve Smith");
  cout << "Number of students in course1: " <<
    course1.getNumberOfStudents() << "\n";
  string* students = course1.getStudents();
  for (int i = 0; i < course1.getNumberOfStudents(); i++)
    cout << students[i] << ", ";
  cout << "\nNumber of students in course2: "
    << course2.getNumberOfStudents() << "\n";
  students = course2.getStudents();
  for (int i = 0; i < course2.getNumberOfStudents(); i++)
    cout << students[i] << ", ";
  return 0;
}
```



```
[huakangleedeMacBook-Pro-7:Downloads huakanglee$ g++ -o TestCourse
TestCourse.cpp Course.cpp
[huakangleedeMacBook-Pro-7:Downloads huakanglee$ ./TestCourse
Number of students in course1: 3
Peter Jones, Brian Smith, Anne Kennedy,
Number of students in course2: 2
Peter Jones,
Steve Smith,
```

Course.h

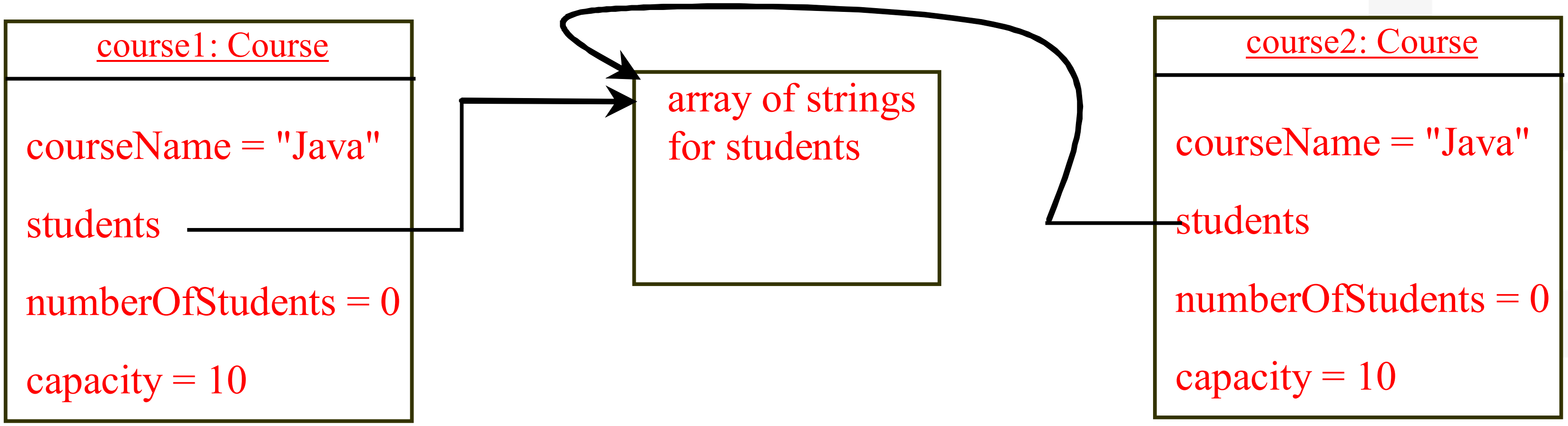
```
#ifndef COURSE_H
#define COURSE_H
#include <string>
using namespace std;

class Course
{
public:
  Course(const string& courseName, int
capacity);
  ~Course();
  string getCourseName() const;
  void addStudent(const string& name);
  void dropStudent(const string& name);
  string* getStudents() const;
  int getNumberOfStudents() const;

private:
  string courseName;
  string* students;
  int numberOfStudents;
  int capacity;
};

#endif
```

Course.cpp

```
#include <iostream>
#include "Course.h"
using namespace std;
Course::Course(const string& courseName, int capacity){
  numberOfStudents = 0;
  this->courseName = courseName;
  this->capacity = capacity;
  students = new string[capacity];
}
Course::~Course(){
  delete [] students;
}
string Course::getCourseName() const{
  return courseName;
}
void Course::addStudent(const string& name){
  tudents[numberOfStudents] = name;
  numberOfStudents++;
}
void Course::dropStudent(const string& name){
  // Left as an exercise
}
string* Course::getStudents() const{
  return students;
}
int Course::getNumberOfStudents() const{
  return numberOfStudents;
}
```

Example 5: Shallow Copy Vs. Deep Copy

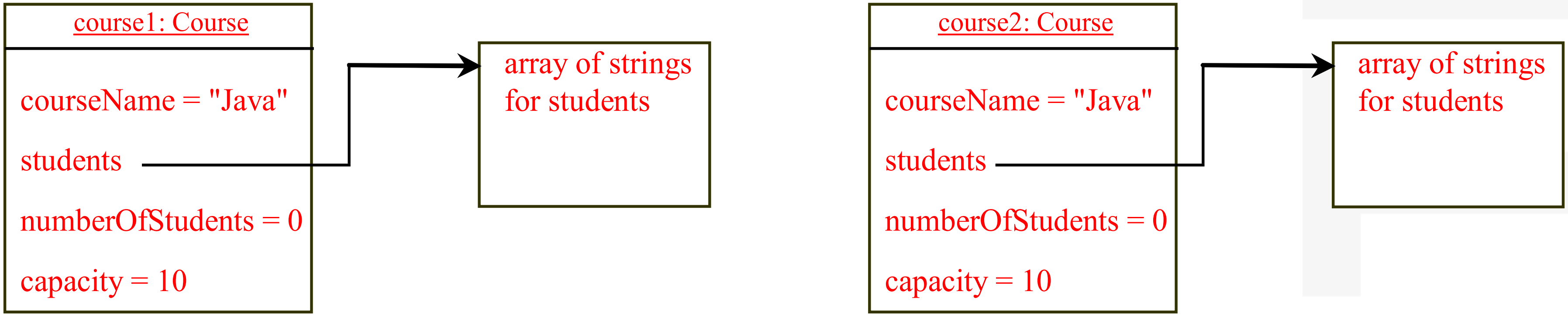
The default copy constructor or assignment operator for copying objects performs a *shallow copy*, rather than a *deep copy*, meaning that if the field is a pointer to some object, the address of the pointer is copied rather than its contents.



Copy course1 to course2. After copying course1 to course2, both course1 and course2 point to the same student.

DEEP COPY

After course1 is copied to course2, the students data field of course1 and course2 point to two different arrays.



CourseWithCustomCopyConstructor.h

```
#ifndef COURSE_H
#define COURSE_H
#include <string>
using namespace std;

class Course
{
public:
  Course(const string& courseName, int capacity);
  ~Course(); // Destructor
  Course(const Course&); // Copy constructor
  string getCourseName() const;
  void addStudent(const string& name);
  void dropStudent(const string& name);
  string* getStudents() const;
  int getNumberOfStudents() const;

private:
  string courseName;
  string* students;
  int numberOfStudents;
  int capacity;
};

#endif
```

ToDo

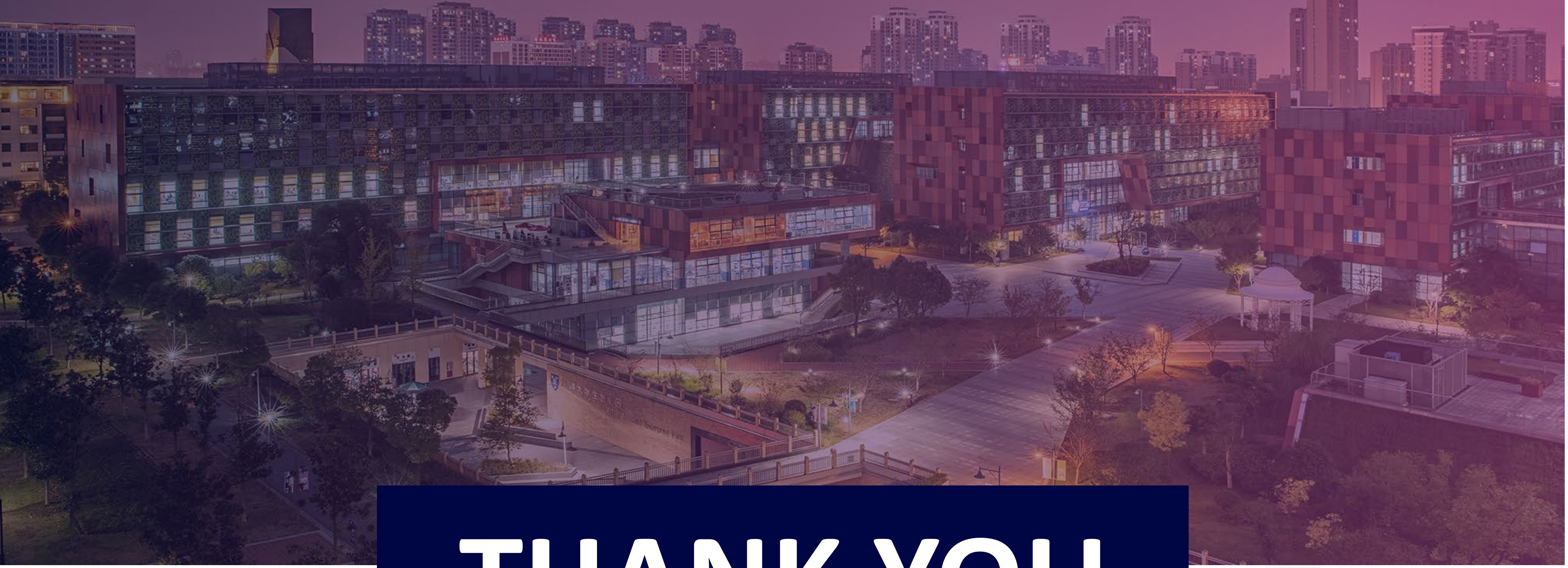
CourseWithCustomCopyConstructor.cpp

CustomCopyConstructorDemo.cpp

CourseWithCustomCopyConstructor.cpp

```cpp
#include <iostream>
#include "CourseWithCustomCopyConstructor.h"
using namespace std;

Course::Course(const string& courseName, int capacity)
{
  numberOfStudents = 0;
  this->courseName = courseName;
  this->capacity = capacity;
  students = new string[capacity];
}

Course::~Course()
{
  delete [] students;
}

string Course::getCourseName() const
{
  return courseName;
}

void Course::addStudent(const string& name)
{
  if (numberOfStudents >= capacity)
  {
    cout << "The maximum size of array exceeded" << endl;
    cout << "Program terminates now" << endl;
    exit(0);
  }

  students[numberOfStudents] = name;
  numberOfStudents++;
}
```

```cpp
void Course::dropStudent(const string& name)
{
  // Left as an exercise
}

string* Course::getStudents() const
{
  return students;
}

int Course::getNumberOfStudents() const
{
  return numberOfStudents;
}

Course::Course(const Course& course) // Copy constructor
{
  courseName = course.courseName;
  numberOfStudents = course.numberOfStudents;
  capacity = course.capacity;
  students = new string[capacity];
}
```



THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

Xi'an Jiaotong-Liverpool University

西交利物浦大學