# PROGRAMMING WITH C++

## LAB 11

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

# Example 1: The This Pointer

Sometimes you need to reference a class's hidden data field in a function. For example, a data field name is often used as the parameter name in a set function for the data field. In this case, you need to reference the hidden data field name in the function in order to set a new value to it. A hidden data field can be accessed by using the **this** keyword, which is a special built-in pointer that references to the calling object.

You can rewrite the Circle class defined in CircleWithPrivateDataFields.h in Chapter 9 using the **this** pointer

## CircleWithThisPointer.cpp

```
#include "CircleWithPrivateDataFields.h"  // Defined in Chapter 9
// Construct a default circle object
Circle::Circle()
{
  radius = 1;
}
// Construct a circle object
Circle::Circle(double radius)
{
  this->radius = radius; // or (*this).radius = radius;
}
// Return the area of this circle
double Circle::getArea()
{
  return radius * radius * 3.14159;
}
// Return the radius of this circle
double Circle::getRadius()
{
  return radius;
}
// Set a new radius
void Circle::setRadius(double radius)
{
  this->radius = (radius >= 0) ? radius : 0;
}
```

# Example 2: Destructors

Destructors are the opposite of constructors. A constructor is invoked when an object is created and a destructor is invoked when the object is destroyed.

Every class has a default destructor if the destructor is not explicitly defined.

Sometimes, it is desirable to implement destructors to perform customized operations.

Destructors are named the same as constructors, but you must put a tilde character (~) in front of it.

Please rewrite the Circle class with Destructor

CircleWithDestructor.h

```
#ifndef CIRCLE_H
#define CIRCLE_H

class Circle
{
public:
  Circle();
  Circle(double);
  ~Circle(); // Destructor
  double getArea() const;
  double getRadius() const;
  void setRadius(double);
  static int getNumberOfObjects();

private:
  double radius;
  static int numberOfObjects;
};

#endif
```

## CircleWithDestructor.cpp

```
#include "CircleWithDestructor.h"

int Circle::numberOfObjects = 0;

// Construct a default circle object
Circle::Circle()
{
  radius = 1;
  numberOfObjects++;
}

// Construct a circle object
Circle::Circle(double radius)
{
  this->radius = radius;
  numberOfObjects++;
}

// Return the area of this circle
double Circle::getArea() const
{
  return radius * radius * 3.14159;
}

// Return the radius of this circle
double Circle::getRadius() const
{
  return radius;
}

// Set a new radius
void Circle::setRadius(double radius)
{
  this->radius = (radius >= 0) ? radius : 0;
}

// Return the number of circle objects
int Circle::getNumberOfObjects()
{
  return numberOfObjects;
}

// Destruct a circle object
Circle::~Circle()
{
  numberOfObjects--;
}
```

## TestCircleWithDestructor.cpp

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

int main()
{
  Circle* pCircle1 = new Circle();
  Circle* pCircle2 = new Circle();
  Circle* pCircle3 = new Circle();

  cout << "Number of circle objects created: "
    << Circle::getNumberOfObjects() << endl;

  delete pCircle1;

  cout << "Number of circle objects created: "
    << Circle::getNumberOfObjects() << endl;

  return 0;
}
```

# Example 3: Copy Constructor

Each class may define several overloaded constructors and one destructor. Additionally, every class has a *copy constructor*. The signature of the copy constructor is:

ClassName(const ClassName&)

For example, the copy constructor for the Circle class is

Circle(const Circle&)

The copy constructor can be used to create an object initialized with another object's data. By default, the copy constructor simply copies each data field in one object to its counterpart in the other object.

Write the program using Circle class with copy constructor.

# CopyConstructorDemo.cpp

```
#include <iostream>
#include "CircleWithDestructor.h" // Defined in Listing 11.11
using namespace std;
int main()
{
  Circle circle1(5);
  Circle circle2(circle1); // Use copy constructor
  cout << "After creating circle2 from circle1:" << endl;
  cout << "\tcircle1.getRadius() returns "
    << circle1.getRadius() << endl;
  cout << "\tcircle2.getRadius() returns "
    << circle2.getRadius() << endl;
  circle1.setRadius(10.5);
  circle2.setRadius(20.5);
  cout << "After modifying circle1 and circle2: " << endl;
  cout << "\tcircle1.getRadius() returns "
    << circle1.getRadius() << endl;
  cout << "\tcircle2.getRadius() returns "
    << circle2.getRadius() << endl;
  return 0;
}
```

```
[huakangleedeMacBook-Pro-7:Downloads huakanglee$ g++ -o CopyConstru]
ctorDemo CopyConstructorDemo.cpp CircleWithDestructor.cpp
[huakangleedeMacBook-Pro-7:Downloads huakanglee$ ./CopyConstructorD]
emo
After creating circle2 from circle1:
        circle1.getRadius() returns 5
        circle2.getRadius() returns 5
After modifying circle1 and circle2:
        circle1.getRadius() returns 10.5
        circle2.getRadius() returns 20.5
```

# Example 4: Implementing The *String* Class (1)

The string class is provided in the C++ library. Provide your own implementation for the following functions (name the new class MyString):

```
MyString();
MyString(const char* cString);
char at(int index) const;
int length() const;
void clear();
bool empty() const;
int compare(const MyString& s) const;
int compare(int index, int n, const MyString& s)
const;
void copy(char s[], int index, int n);
char* data() const;
int find(char ch) const;
int find(char ch, int index) const;
int find(const MyString& s, int index) const;
```

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

const int MAXIMUM_SIZE = 100; // Maximum string size 100

class MyString
{
public:
  MyString()
  {
    size = 0;
  }

  MyString(const char* chars)
  {
    int i = 0;
    for (; chars[i] != '\0'; i++)
      value[i] = chars[i];

    size = i;
  }

  char at(int index) const
  {
    return value[index];
  }

  int length() const
  {
    return size;
  }

  void clear()
  {
    size = 0;
  }

  bool empty() const
  {
    return size == 0;
  }

  int compare(const MyString& s) const
  {
    return compare(0, size, s);
  }

  int compare(int index, int n, MyString s) const
  {
    char* s1 = data();
    char* s2 = s.data();

    return strcmp(s1, s2);
  }

  void copy(char s[], int index, int n)
  {
    int i = 0;
    for (; i < size && i < n; i++)
      s[i] = value[i + index];

    i = i + 1;
    s[i] = '\0';
  }

  char* data() const
  {
    char * temp = new char[size + 1];
    for (int i = 0; i < size; i++)
      temp[i] = value[i];

    temp[size] = '\0';

    return temp;
  }

  int find(char ch) const
  {
    return find(ch, 0);
  }

  int find(char ch, int index) const
  {
    for (int i = index; i < size; i++)
      if (value[i] == ch) return i;

    return -1;
  }

  int find(const MyString& s, int index)
const
  {
    return -1;
  }

private:
  char value[MAXIMUM_SIZE];
  int size;
}; // Must place a semicolon here
```

```
int main()
{
  MyString stringObject1("abc abc");
  MyString stringObject2("efgth");

  for (int i = 0; i < stringObject1.length(); i++)
    cout << stringObject1.at(i);

  cout << endl;

  cout << stringObject1.length() << endl;

  cout << stringObject1.empty() << endl;

  cout << stringObject1.compare(stringObject2) << endl;

  cout << stringObject1.find('a') << endl;
  cout << stringObject1.find('a', 4) << endl;
  cout << stringObject1.find('a', 6) << endl;

  char* temp = stringObject1.data();
  cout << temp << endl;

  stringObject2.clear();
  cout << stringObject2.length() << endl;

  return 0;
}
```

# Example 5: Implementing The *String* Class (2)

The string class is provided in the C++ library. Provide your own implementation for the following functions (name the new class MyString):

```
MyString(const char ch, int size);
MyString(const char chars[], int size);
MyString append(const MyString& s);
MyString append(const MyString& s, int index,
int n);
MyString append(int n, char ch);
MyString assign(const char* chars);
MyString assign(const MyString& s, int index,
int n);
MyString assign(const MyString& s, int n);
MyString assign(int n, char ch);
MyString substr(int index, int n) const;
MyString substr(int index) const;
MyString erase(int index, int n);
```

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

const int MAXIMUM_SIZE = 100; // Maximum string size 100

class MyString
{
public:
  MyString(char ch, int n)
  {
    for (int i = 0; i < n; i++)
      value[i] = ch;

    size = n;
  }

  MyString(const char chars[], int size)
  {
    for (int i = 0; i < size; i++)
      value[i] = chars[i];

    this->size = size;
  }
```

```
  MyString append(const MyString& s)
  {
    for (int i = 0; i < s.length(); i++)
      value[size + i] = s.at(i);

    size += s.length();

    return MyString(value, size);
  }

  MyString append(const MyString& s, int index, int n)
  {
    for (int i = 0; i < s.length() && i < n; i++)
      value[size + i] = s.at(index + i);

    size += s.length();

    return MyString(value, size);
  }

  MyString append(int n, char ch)
  {
    for (int i = size; i < size + n; i++)
      value[i] = ch;

    size += n;

    return MyString(value, size);
  }
```

```
  MyString assign(const char* chars)
  {
    int i;
    for (i = 0; chars[i] != '\0'; i++)
      value[i] = chars[i];

    size = i;

    return MyString(value, size);
  }

  MyString assign(const MyString& s, int index, int n
  {
    int i;
    for (i = 0; i < n; i++)
      value[i] = s.at(index + i);

    size = n;

    return MyString(value, size);
  }

  MyString assign(const MyString& s, int n)
  {
    return assign(s, 0, n);
  }

  MyString assign(int n, char ch)
  {
    for (int i = 0; i < n; i++)
      value[i] = ch;

    size = n;

    return MyString(value, size);
  }
```

```
char at(int index) const
 {
  return value[index];
 }

 int length() const
 {
  return size;
 }

 MyString erase(int index, int n)
 {
  char temp[MAXIMUM_SIZE];
  for (int i = 0; i < size && i < index; i++)
    temp[i] = value[i];

  return MyString(temp, size - n);
 }

 char* data()
 {
  char * temp = new char[size + 1];
  for (int i = 0; i < size; i++)
    temp[i] = value[i];

  temp[size] = '\0';

  return temp;
 }
```

```
MyString substr(int index, int n)
 {
  char * temp = new char[n];
  for (int i = 0; i < size && i < n; i++)
    temp[i] = value[i + index];

  return MyString(temp, n);
 }

 MyString substr(int index)
 {
  return substr(index, size - index - 1);
 }

 void swap(MyString s)
 {
  char * temp = s.data();
  int tempSize = s.length();

  s.assign(value);

  // Copy temp to value
  for (int i = 0; i < tempSize; i++)
    value[i] = temp[i];
  size = tempSize;
 }

private:
 char value[MAXIMUM_SIZE];
 int size;
}; // Must place a semicolon here
```

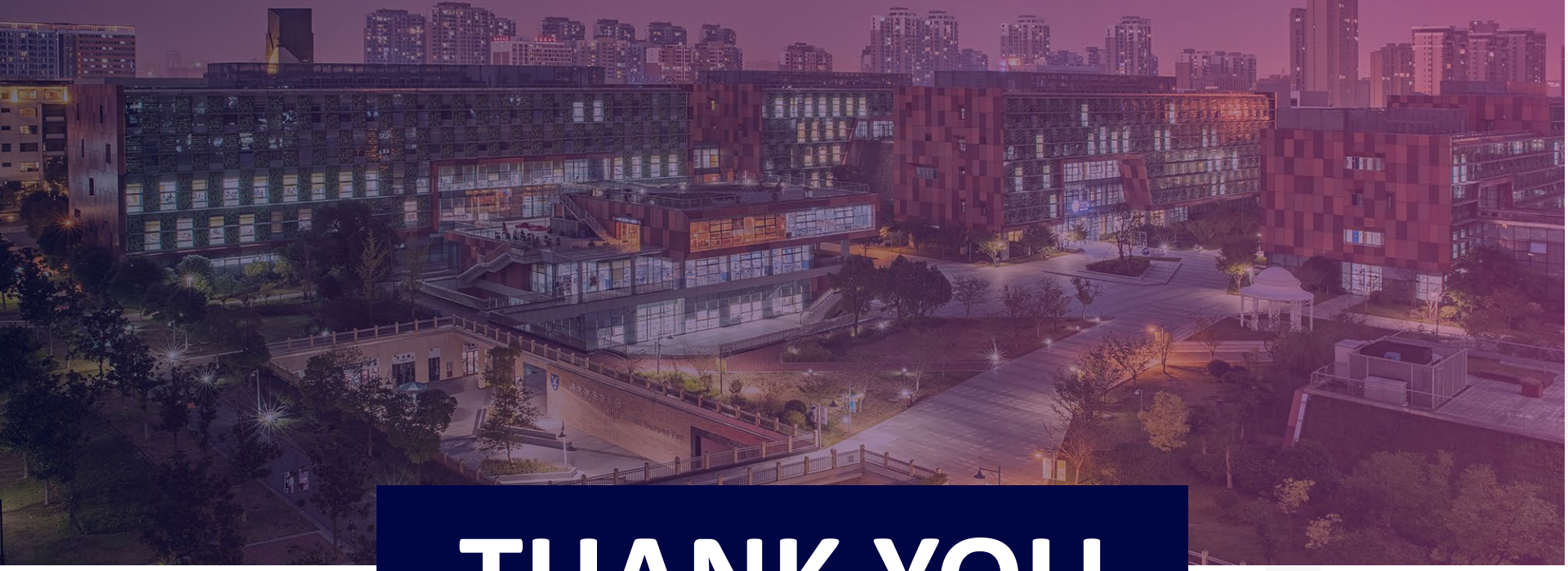
```
int main()
{
 char s[] = {'a', 'b', 'c', 'd', 'e'};
 MyString stringObject1(s, 3);
 MyString stringObject2(s, 3);

 stringObject1.append(stringObject2);

 for (int i = 0; i < stringObject1.length(); i++)
 {
  cout << stringObject1.at(i);
 }

 cout << endl;

 return 0;
}
```



# THANK YOU

## VISIT US

WWW.XJTLU.EDU.CN

## FOLLOW US

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