Introduction to Programming with C++

Chieh-Sen (Jason) Huang

Department of Applied Mathematics

National Sun Yat-sen University

INTRODUCTION TO
PROGRAMMING
WITH
C++
Third Edition

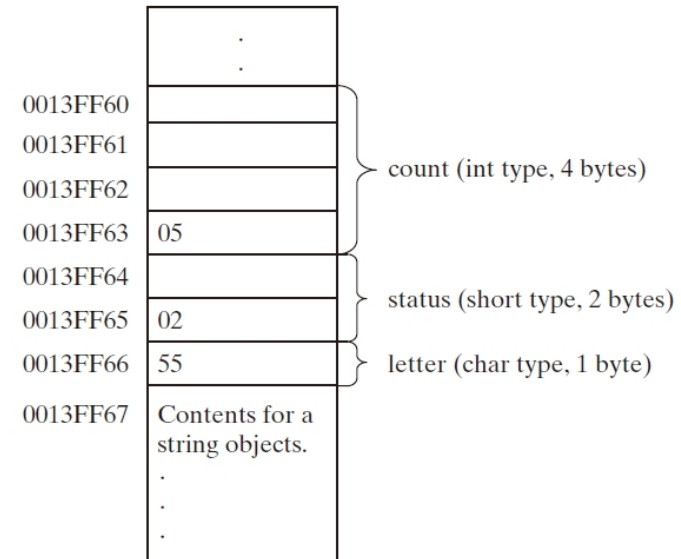Contents are based on book by Y. Daniel Liang

Pointers

- Pointer is one of the most powerful features in C++. It is the heart and soul of the C++ programming language. Many of the C++ language features and libraries are built using pointers.

- To allocate and release the memory for variables on the fly at runtime, it could be accomplished using pointers.

- Pointer variables are also known as pointers. You can use a pointer to **reference** the address of an array, an object, or any variable.

- A pointer variable holds the **memory address**. Through the pointer, you can use the **dereference operator** * to access the actual value at a specific memory location.

- Normally, a variable contains a data value e.g., an integer, a floating-point value, and a character.

- However, a pointer contains the memory address of a variable that in turn contains a data value.

Pointers

```
int count = 5;
short status = 2;
char letter = 'A';
```

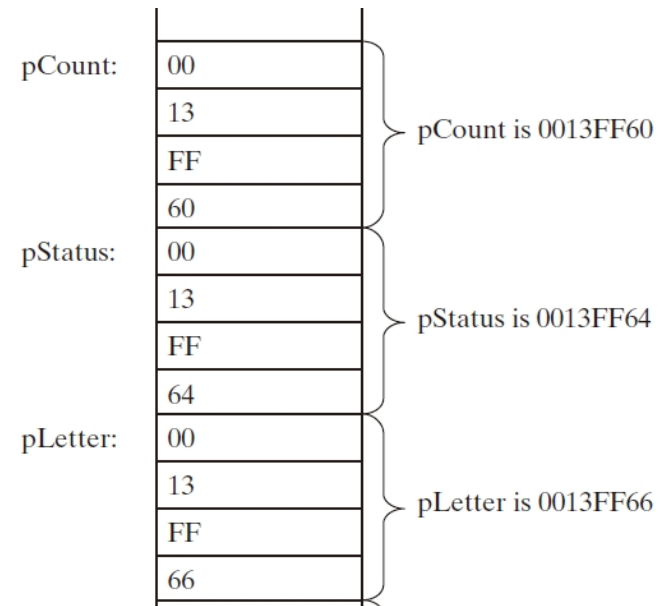- Note that the ASCII code for 'A' is hexadecimal 55.

Address	Value	
	.	
	.	
0013FF60		count (int type, 4 bytes)
0013FF61		
0013FF62		
0013FF63	05	
0013FF64		status (short type, 2 bytes)
0013FF65	02	
0013FF66	55	letter (char type, 1 byte)
0013FF67	Contents for a string objects.	

- Each variable being declared as a pointer must be preceded by an **asterisk (*)**.

- Syntax

```
dataType* pVarName;

int* pCount;
//pointer to an int.
short* pStatus;
char* pLetter;
```
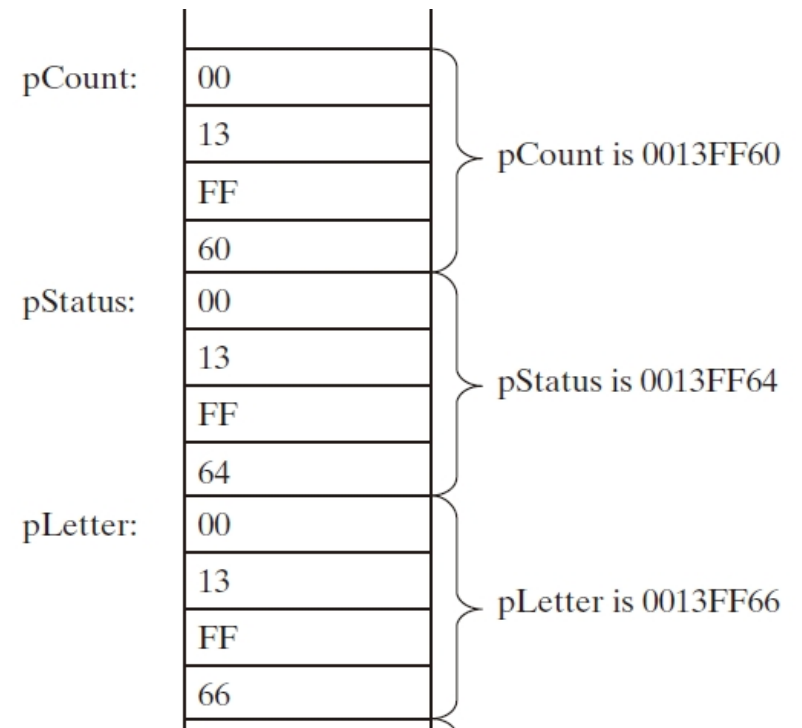
	Value	
pCount:	00	pCount is 0013FF60
	13	
	FF	
	60	
pStatus:	00	pStatus is 0013FF64
	13	
	FF	
	64	
pLetter:	00	pLetter is 0013FF66
	13	
	FF	
	66	

- Pointers are all 4 bytes.

Pointers

- We assign the address of a variable to a pointer as following

  `pCount = &count;`

- The **ampersand (&)** symbol is called the **address operator** when placed in front of a variable. It is a unary operator that returns the **variable's address**.

- You may pronounce **&count** as **the address of count**.

	.	
	.	
0013FF60		
0013FF61		count (int type, 4 bytes)
0013FF62		
0013FF63	05	
0013FF64		status (short type, 2 bytes)
0013FF65	02	
0013FF66	55	letter (char type, 1 byte)
0013FF67	Contents for a string objects.	
	.	
	.	

pCount: 00 / 13 / FF / 60 — pCount is 0013FF60

pStatus: 00 / 13 / FF / 64 — pStatus is 0013FF64

pLetter: 00 / 13 / FF / 66 — pLetter is 0013FF66
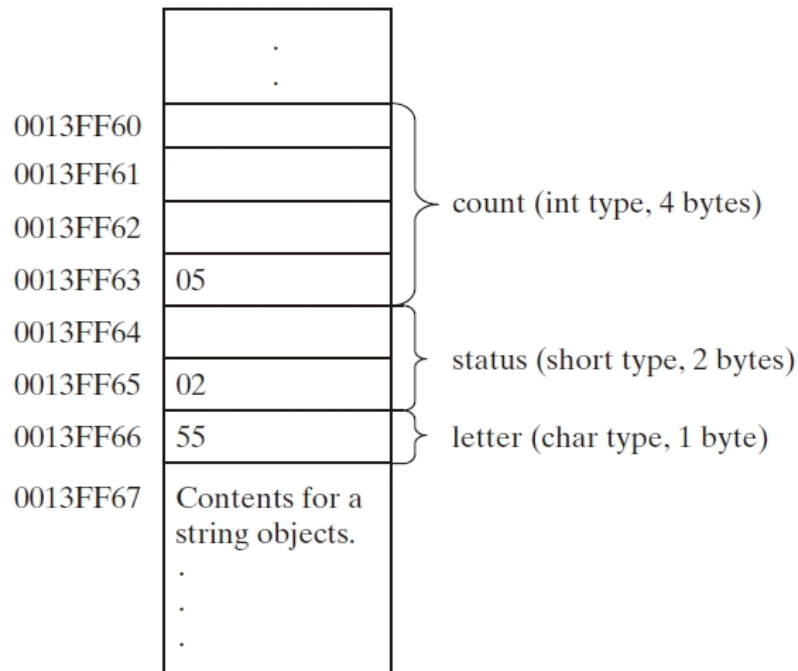
Pointers

```
\\List11_1.cpp
int count = 5;          \\ declare variable
int* pCount = &count; \\ declare pointer, assign &count to pCount
                       \\ i.e. we point pCount to count
cout << "The value of count is "   << count << endl;
cout << "The address of count is " << &count << endl;
cout << "The value of pCount is "  << pCount << endl;
cout << "The address of pCount is "<< &pCount << endl;
cout << "The value of pCount pointed is "
                                   << *pCount << endl;
-----------------------------------------------------------
The value of count is 5
The address of count is 0013FF60
The value of pCount is 0013FF60
The address of pCount is 0019CF12  \\ other location
The value of pCount pointed is 5
```

- Referencing a value through a pointer is often called **indirection**.
```
*pointer
```

Pointers

- The **asterisk (*)** used in the preceding statement is known as the **indirection operator** or **dereference operator** (dereference means indirect reference).

- When a pointer is dereferenced, the value at the address stored in the pointer is retrieved.

- You may pronounce ***pCount** as the **value indirectly pointed by pCount**, or simply **value pointed by pCount**.

- Pointers are variables. So, the naming conventions for variables are applied to pointers.

- We have named pointers with prefix p, such as pCount and pArea. However, it is impossible to enforce this convention.

  In-Class Exercise: Mimic List11_1.cpp and declare an double pointer. Find the corresponding values of this pointer as in List11_1.cpp.

- **An array name is actually a pointer**.

Pointers

- Like a local variable, a local pointer is assigned an arbitrary value if you don't initialize it.

- A pointer may be initialized to 0, which is a special value to indicate that the pointer points to nothing.

- To prevent errors, you should always initialize pointers.

- A number of C++ libraries including <iostream> define **NULL** as a constant with value 0. It is more descriptive to use NULL than 0.

- You can declare an int pointer using the following syntax

```
int* p; // p is an integer pointer

int *p; // p is a pointer pointed to integer

int * p;// Well, I can't explain this!!
```

Pointers
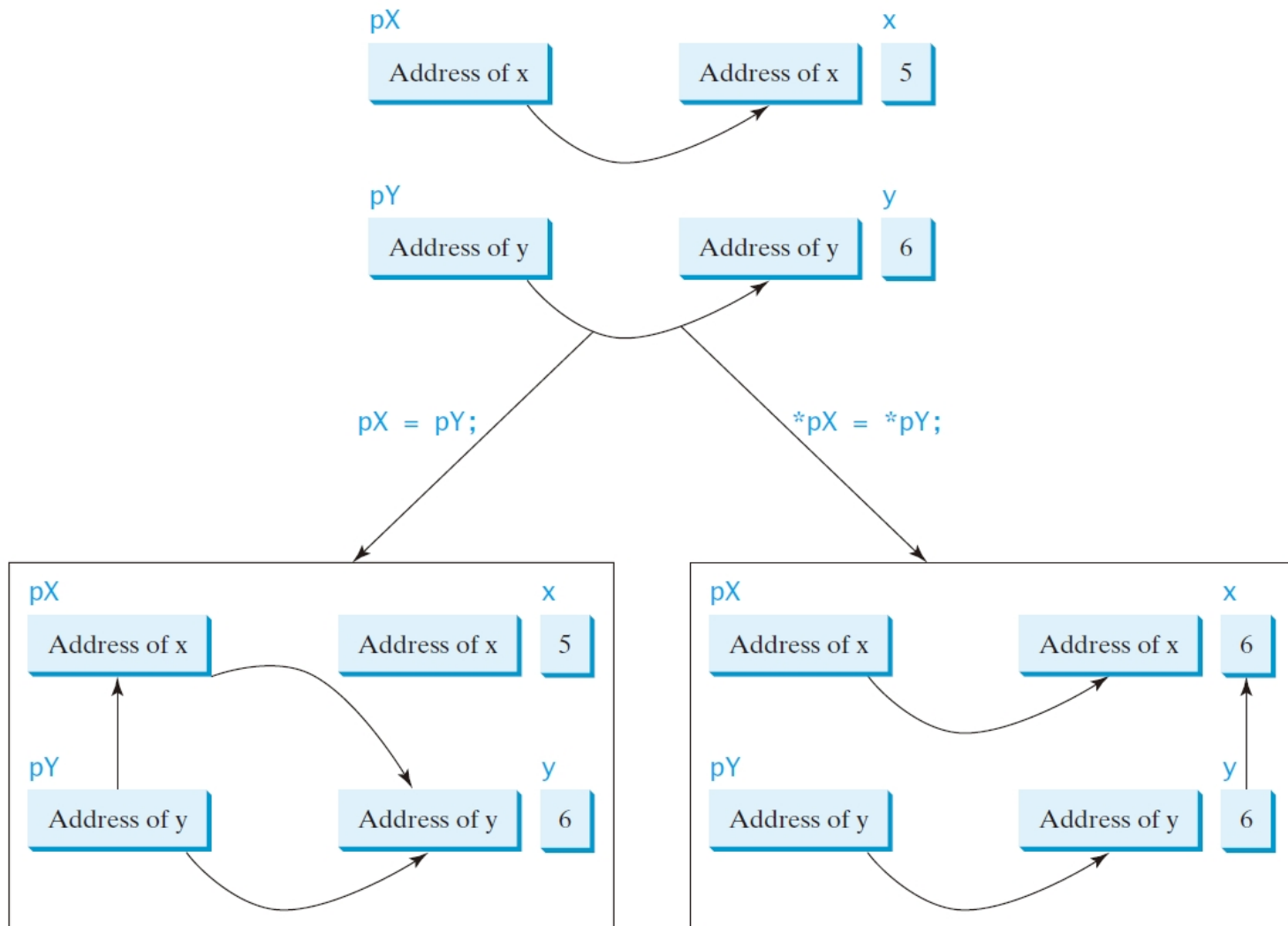
Dynamical Array

- C++ supports dynamic memory allocation, which enables you to allocate persistent storage dynamically.

- The memory is created using the **new operator**.
  ```
  int* p = new int[4];

    cout << "Enter the size of the array: ";
    int asize;
    cin >> asize;
    int* a = new int[asize];
  ```

- **new int(size)** tells the computer to allocate memory space for an int array with the specified number of elements.

- The address of the array is assigned to pointer **a**.

- The array created using the new operator is also known as a dynamic array.

- Note that when you create a regular array, its size must be known at compile time.

Dynamical Array

```
cout << " Enter the size of the array: "
cin >> asize; \\List11_1a.cpp
a = new int[asize];

for(i = 0; i < asize; i++)
   a[i] = i;

for(i=0; i<asize; i++)
   cout<<" a["<< i <<"] = "<< a[i] <<'\n';

delete[] a;
-----------------------------------------------------
 Enter the size of the array: 5
a[0] = 0
a[1] = 1
a[2] = 2
a[3] = 3
a[4] = 4
```

In-Class Exercise: Rerite the code of bubble sort using dynamical array.

Dynamical Array

- The word **delete** is a keyword in C++. If the memory is allocated for an array, the () symbol must be placed between the delete keyword and the pointer to the array to release the memory properly.

- Use the delete keyword only with the pointer that points to the memory created by the new operator.

```
int* p = &x;
delete p; // This is wrong
```

Homework 11.1: Programming exercise: 11.1, 11.5.

- Goto Chapter 8.

- Back from Chapter 10.

Creating and Accessing Dynamic Objects

- To create an object dynamically, invoke the constructor for the object using the syntax `new ClassName(arguments)`

```
ClassName* pObject = new ClassName;
```

creates an object using the no-arg constructor and assigns the object address to the pointer.

```
ClassName* pObject = new ClassName(arguments);
```

creates an object using the constructor with arguments and assigns the object address to the pointer.

Creating and Accessing Dynamic Objects

- To access object members via a pointer, you must dereference the pointer and use the dot (.) operator to object's members.

```
string* p = new string("abcdefg"); //List11_3a.cpp
cout << "The first three characters in the string are "
     << (*p).substr(0, 3) << endl;
cout << "The length of the string is "
     << (*p).length() << endl;
--------------------------------------------------

The first three characters in the string are abc
The length of the string is 7
```

- To accesse object members from a pointer:
arrow operator (->), which is a dash (-) immediately followed by the greater than (>) symbol.

```
cout << "The first three characters in the string are "
     << p->substr(0, 3) << endl;
cout << "The length of the string is "
     << p->length() << endl;
--------------------------------------------------

The first three characters in the string are abc
The length of the string is 7
```

Dynamical Object Array

- **new operator** allows us to dynamic allocate array of objects.

```cpp
int N; //List11_4a.cpp
cout << "Input number of Circles:";
cin >> N;
Circle* pC = new Circle[N];
for( int i = 0; i < N; i++){
   pC[i].radius = i*1.0;
   cout << "The area of the pC[" << i << "] is "
        << (*(pC + i)).getArea() << endl;
//The same as pC[i].getArea()
    cout << "The area of the pC[" << i << "] is "
        << (pC + i)->getArea() << endl;
}
------------------------------------------------------------
Input number of Circles:2
The area of the pC[0] is 0
The area of the pC[0] is 0
The area of the pC[1] is 3.14159
The area of the pC[1] is 3.14159
```

In-Class Exercise: Write a code to allocate Vector array of length N.

The this Pointer

- The **this** pointer points to the calling object itself.

- A hidden data field can be accessed by using the this keyword, which is a special built-in pointer that references the calling object.

```
Circle::Circle(double radius)
{                             // CircleWithThisPointer.cpp
  this->radius = radius; // or (*this).radius = radius;
}
void Circle::setRadius(double radius)
{
  this->radius = (radius >= 0) ? radius : 0;
}
```

- The parameter name radius in the constructor is a local variable. To reference the data field radius in the object, you have to use this->radius.
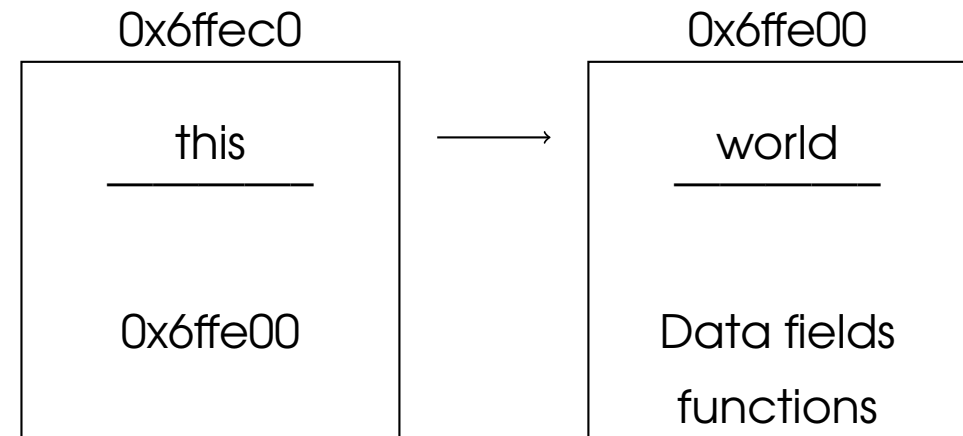
The this Pointer

```cpp
class Str {      \\ List11_2a.cpp
public:
  void show() { cout << str << endl; }
  void set(char * ptr) { str = ptr; }
  void showthis() { cout << this << endl; }
  void showaddressthis() {
         Str * const &add = this;
         cout << &add << endl;}
private:
  char * str;
};

int main()
{
  Str hello, world;
  hello.set("Hello");
  world.set("World!");
  cout << "The address of hello object "
       << &hello << endl;
  cout << "The address of world object "
       << &world << endl;;
  world.showthis();
  world.showaddressthis();
}
-------------------------------------------------
The address of hello object 0x6ffe10
The address of world object 0x6ffe00
0x6ffe00
0x6ffec0
```

```
            0x6ffec0                        0x6ffe00
  +------------------------+      +------------------------+
  |                        |      |                        |
  |          this          |      |         world          |
  |        _____        | ---> |        _____        |
  |                        |      |                        |
  |                        |      |                        |
  |        0x6ffe00        |      |       Data fields       |
  |                        |      |                        |
  |                        |      |       functions        |
  +------------------------+      +------------------------+
```

Destructors

- Every class has a destructor, which is called automatically when an object is deleted.

- Destructors are the opposite of constructors.

- Destructors are named the same as constructors, but you must put a tilde character ($\sim$).

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

- See List11_13.cpp

Copy Constructors

- Every class has a **copy constructor**, which can be used to create an object initialized with the data of another object of the same class.

```
ClassName(const ClassName&)
Circle(const Circle&)
```

- A default copy constructor is provided for each class implicitly, if it is not defined explicitly. The default copy constructor simply copies each data field in one object to its counterpart in the other object.

```
Circle circle1(5);
Circle circle2(circle1); // Use copy constructor
```

Homework 11.2: Programming exercise: 11.8.