Object Oriented Programming
CS250

Dr Ahmed Rafat Abas

Computer Science Dept, Faculty of
Computers & Informatics, Zagazig University

arabas@zu.edu.eg

http://www.arsaliem.faculty.zu.edu.eg

Pointers and Dynamic Allocation

Chapter 7

Dr Ahmed Rafat

6.1 Introduction

- Pointers and dynamic memory allocation allow us to build dynamic data structures that adapt to the application needs while a program is running.

- Pointers must also be used with care to avoid corrupting memory.

6.2 Pointers to Constants

6.2.1 Pointer to a constant

- A pointer to a constant is a pointer that is declared in such a way that the object it points to cannot be modified via that pointer.

Example:

```
int n = 0;
const int * cp = &n;
*cp = 30;       // error
n = 30;  // but this is ok
```

6.2.2 Const-Qualified Pointers

- Sometimes it is required to prevent the contents of a pointer itself from being modified.

Example:

```
char message[80];
char * const sp = message;
sp++;              // error
scrcpy(sp, "A new message");     // ok
```

6.2.3 Const-Qualified Pointers to constants

- If neither the pointer nor the data it points to should be changed, we can use **const** twice to create a const-qualified pointer to a constant:

Example:

```
char message[80];
const char * const sp = message;
sp++;          // error
strcpy(sp, "A new message");      // error
```

- To summarize, there are four possible ways that we could have declared a pointer to message.

Example:

```
char *p1 = message;
char * const p2 = message;
const char *p3 = message;
const char * const p4 = message;
```

6.3 Functions Returning Pointers to Constants

- When a function returns a pointer to a constant, the variable receiving the return value must also be a pointer to a constant.

Example:

```
const char * GetName() const;
//…
char * ncName = S.GetName();   // error
const char * cName = S.GetName();    // ok
```

6.4 Pointers to Array Elements

- When memory is allocated for a pointer, *C++* keeps track of the type of object to which the pointer points, which includes its implementation size.

Example:

```
float flist[] = {10.5, 13.2, 4, 9.6};  // array of float
float * fp = flist;                    // points to flist[0]
fp++;                                  // points to flist[l]
```

6.5 Void Pointers

- *C/C++* programmers use the void pointer type to achieve flexibility of types.
- Any pointer can be assigned to a void pointer.

Example:

```
int * p;
void * v = p;      // ok
p = (int *) v;     // cast required
```

6.6 References to Pointers

- A pointer can be passed to a function by reference in cases where the pointer must be modified by the function.

Example:

```
void FindNext(char * & p, char delim)
{
   while( *p && (*p != delim))
   p++;
}
```

6.7 Implicit Conversion of Derived Pointers to Base Pointers

- a base type pointer can point at either a base object or at a derived object.

Example:

```
point3D center;
point * P = &center;
```

- a reference to an object of a derived class can be implicitly converted to a reference of its base type

Example:

```
Point3D * cp = new Point3D;
Point * p;
p = cp;
```

- This type of pointer conversion usually takes place when a pointer of a derived type is passed to a function whose parameter is of a base type.

Example:

```
void CalcTuition( Student * sp ){//…}
//…
GraduateStudent * gp=new GraduateStudent;
CalcTution(gp);
```

- Our example using pointers would also have worked with references to Student and GraduateStudent objects:

Example:

```
void CalcTuition( Student & S )
// S refers to a Student or GraduateStudent
…
GraduateStudent aGrad;
CalcTuition( aGrad );
```

6.8 Fixed and Dynamic Memory Allocation

- Fixed allocation applies to an object whose size is determined at compile time.
- Dynamic allocation is the process of allocating memory for objects at run time.
- Objects that use dynamic allocation need not have a specified size at compile time.
- A dynamic object's lifetime continues until either:
  - the program ends, or
  - the object is explicitly deallocated.

6.9 The New and Delete Operators

- The **new** operator allocates a block of storage and returns the address of the storage.
- When an array is allocated, the program's runtime system keeps track of the array's size.
- Storage is deallocated by using the **delete** operator but the pointer itself is not deleted.

Example:

```
int * p = new int;
int * array = new int[50];
//…
delete p;                 // a single object
delete [] array;          // an array
```

6.10 Storage Duration and Pointers

- A pointer may have automatic storage duration, while the memory it addresses has dynamic duration.
- In this case, the pointer can go out of scope while the dynamic memory remains allocated:

Example:

```
if( a > b )
{
   float* fp = new float;
   //...
}
```

- the storage is left unavailable when fp goes out of scope at the end of the block.
- This is a common programming error called a memory leak.

6.11 Pointers to Class Objects

- An object's constructor executes immediately after the **new** operator successfully allocates storage for the object.
- If the **new** operator fails to allocate storage, the constructor is not executed and the **new** operator returns a null address (0).

Example:

```
Student * p;            // declare a pointer
p = new Student;        // allocate storage
if( !p )
// could not create a Student
```

- When we create an array of class objects, the class's default constructor is called for each member of the array:

Example:

```
Point * figure = new Point[l0];
// Calls Point() ten times,
```

- The delete operator, when applied to a pointer containing the address of a class object, will cause the object's destructor to be called.

Example:

```
delete [] figure;   //the class destructor will
// be called for each member of the array
```

6.12 Arrays and Dynamic Allocation

- **One-Dimensional Arrays**

```
float  *  myArray = new float [1000] ;
```

- **Two-Dimensional Arrays**

```
int ( * table ) [10] = new int[3] [10];
```

or

```
Int * table[3];          // defining array of pointers to rows
For ( i=0; i<3; i++ )
      table[i] = new int[10]; //dynamic allocation of
                              // memory for each row.
```

6.13 Constructors and Destructors

- Classes containing pointer data members frequently use the **new** operator in their constructors to allocate storage.
- The delete operator should be used in the destructor function to release memory allocated in the constructor function.

6.14 Copy Constructors

- A *copy constructor* is a special-purpose constructor that makes a duplicate copy of an object of the same type.

Example:

```
Student A;     // ( Student A is initialized... )
Student B( A );
Student C = A;
Student Modify_Student ( Student A );
```

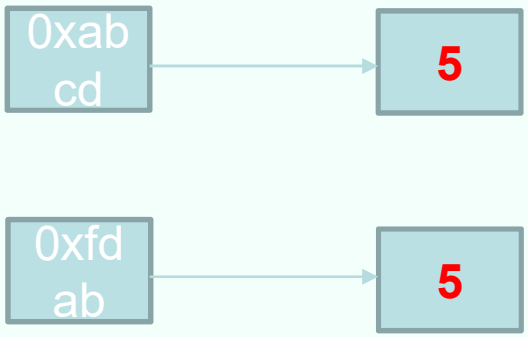
- A copy constructor is invoked automatically when a temporary object is constructed from an existing one.

Example: if a Student were passed to a function that used a value parameter (rather than a reference), the compiler would construct a temporary copy of the passed argument, aStudent:

```
void RegisterStudent( Student S );
…
Student aStudent;
RegisterStudent( aStudent );
```

6.15 Cross-Linking

- Two pointers to the same storage location is dangerous and can lead to serious runtime errors.
- To prevent that from happening, always provide a copy constructor for a class containing pointers.


0xab
cd
5
0xfd
ab
5

6.16 Common Pointer errors

1. Encapsulating the new Operator

- **Avoid the use of the new operator in user code**, that is, outside classes.
- Memory allocation errors can be introduced into programs when you allocate storage and later forget to release it.

2. Uninitialized Pointer

- **Deleting storage indicated by a pointer that was never initialized is a serious error.** Such as deleting the same pointer twice.
- But deleting a null pointer is guaranteed in standard C++ not to have any effect.
- **It is recommend to set a pointer** to null when it does not point to actual data.

3. Null Pointer Assignment

- **A common programming error is made when attempting to dereference a null pointer and use it to access memory.**
- **Memory may be corrupted, causing the program to behave strangely.**

Example:

```
if ( Lp ! = 0 )
        *Lp = 5000;          // assign a value
```

4. Dangling Pointer

- **A dangling pointer is a pointer that no longer contains the address of allocated storage.**
- **One way to cause this error is to create a local variable and then try to use the variable's address outside the block.**

Example:

```
char * Input ( )
{
        char buffer[128];
        cout « "Enter last name: ";
        cin.getline( buffer, 128 );
        return buffer;
}
```