CSC112 Spring 2011
Fundamentals of Computer Science
Summary of Pointers

What is is a C++ pointer ?

A pointer is a variable that holds an address (i.e., a location in computer memory) containing an object of some underlying data type.

How to declare a pointer

The syntax for declaring a pointer is illustrated below.

Note that "Data type" can be something more complicated than just one of the basic types.

```
//   Examples:
//
    int    * ip ; // Variable ip is a pointer to an integer.
    char   * cp ; // Variable cp is a pointer to a character.
    double * dp ; // Variable dp is a pointer to a double.
    int   ** pt ; // Variable pt is a pointer to a pointer to an int.
```

How to initialize a pointer

Using a pointer without initializing it is likely to create errors. There are several ways in which a pointer can be initialized. Here are some common ways:

1. A pointer may be assigned the address of an existing object using the unary `&` operator.

```
//   Example:
//
    int * p ;
    int b   ;

    b = 99  ;   // b is an ordinary integer variable.
    p = & b ;   // p points to (holds the address of) b.
```

2. A pointer may be assigned a value using keyword `new`. The C++ keyword `new` is useful for creating single objects, or an array of objects. The keyword `new` allocates memory for a new object. For example (a single object):

```
//   Example of allocating a single object:
//
    int * p ;

    p = new int ;   // p points to memory containing a single int.
```

   For example (an array of objects):

```
//   Example of allocating an array:
//
    int * a ;
    int n ;

    cin >> n ;          // a points to a block of memory
    a = new int [n] ;   // containing n integers.
```

What operations apply to pointers ?

1. Assignment. One pointer can be copied to another.
2. Pointer dereference. Follow the pointer to the indicated data.
3. Addition and subtraction. Pointer arithmetic can be used to refer to an offset within a block of memory.
4. Array subscripting. Array subscripting can be thought of as a combination of pointer arithmetic (adding an offset to a pointer) and pointer dereference.
5. Parameter passing. The pointer itself may be passed to a function by value. But, if the function uses pointer indirection to store data into the indicated memory, the caller will also have access to the stored data.
6. Pointer coercion. The underlying data type to which a pointer refers may be changed through pointer coercion. The primary use of pointer coercion is to create parameter type compatibility for function calls, such as the `read()` and `write()` functions of `ifstream` and `ofstream` objects.

Examples of the above operations

1. Pointer assignment

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

//  Example to show pointer assignment (the = operator).
//
int main()
{
   int b ;
   int * p, * q ;

   b = 5  ;  // Ordinary assignment to an integer .
   p = &b ;  // p is assigned the address of b.
   q = p ;  //  q now points to the same memory location as p.

   *q = 7 ;  // Changing the contents of memory pointed to by q.

   cout << b << endl ;   // Print the value of b.  Get 7, not 5.
}
```

```
----------------------  Sample Session -----------------------
atlas% g++ assign_ptr.cc
atlas% a.out
7
```

2. Pointer dereference

Pointer dereferencing may be used 1) in an expression, or 2) on the left hand side of an assignment statement. Here is an example of both types of use:

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

//  Example to illustrate pointer dereference.
//
int main()
{
   int b, c ;
   int * p  ;

   b = 5  ;  // Ordinary assignment to an integer .
   p = &b ;  // p is assigned the address of b.

   c = *p + 10  ;  // Pointer dereference on the right side of "=".

   cout << "c = " << c << endl ;  // Print c, get 15

   *p = 7 ;  // Pointer dereference on the left side of "=".
             // This changes the contents of memory indicated by p.

   cout << "b = " << b << endl ;  // Print the value of b.  Get 7, not 5.
}

---------------------- Sample Session ----------------------
atlas% g++ deref.cc
atlas% a.out
c = 15
b = 7
```

3. Pointer arithmetic

Pointer arithmetic can be used to refer to a place in memory which we specify as a reference point (the place indicated by the pointer) plus an offset (some place "down the road"). In the following example, we add a pointer and an integer. **Note: When adding a pointer to an integer , the integer is automatically adjusted to reflect the size of the underlying base type.**

Often, we use pointers to manage dynamically allocated arrays. In this example, we see that a pointer can refer to a fixed-length array.

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

//  Example to illustrate pointer arithmetic using character strings.
//
int main()
{
   char s[ 80 ] ;  // Fixed length array to hold a small string.
   char * p ;      // Pointer.
   int n    ;      // Counter.

   cin >> s ;  // Read a character string.
   p = s    ;  // Initialize a pointer to the beginning of a string.
               // Notice that arrays and pointers are compatible.
   n = 0    ;  // Start counting with zero.

   // In C++, strings represented by arrays of characters use the
   // null character (ASCII zero) to mark the end of a string.
   // We use the syntax '\0' to denote the null character.

   // This loop keeps counting upward until we find the null character.
   // Notice the value of n is added to the pointer before the dereference.

   while ( *(p+n) != '\0' ) {
      n++ ;
   }

   // Print the length of the string.
   cout << "'" << s << "'" << " is length " << n << endl ;
}

------------------------- Sample Session -------------------------
atlas% g++ arith.cc
atlas% a.out
Hello
'Hello' is length 5
```

4. Array subscripting with pointers

In this example, we further illustrate the compatibility between arrays and pointers. Once a pointer to a block of memory is initialized, we can use subscript notation `[]` to refer to places in memory which are offset from the beginning of the block.

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

//  Example to illustrate array subscripting using character strings.
//
int main()
{
   char s[ 80 ] ;  // Fixed length array to hold a small string.
   char * p ;      // Pointer.
   int n    ;      // Counter.

   cin >> s ;  // Read a character string.
   p = s    ;  // Initialize a pointer to the beginning of a string.
               // Notice that pointers and arrays are compatible.
   n = 0    ;  // Start counting with zero.

   // In C++, strings represented by arrays of characters use the
   // null character (ASCII zero) to mark the end of a string.
   // We use the syntax '\0' to denote the null character.

   // This loop keeps counting upward until we find the null character.
   // Notice the value of n is added to the pointer before the dereference.

   while ( p[n] != '\0' ) {    // Notice we can use [] on a pointer.
      n++ ;
   }

   // Print the length of the string.

   cout << "The word '" << s << "'" << " is length " << n << endl ;

}

---------------------------- Sample Session -----------------------------
atlas% g++ subscript.cc
atlas% a.out
armadillo
The word 'armadillo' is length 9
```

5. Passing pointers as function parameters

When a pointer is used as a function parameter, it gives the function access to memory which is also accessible by the caller. Here is an example.

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

//  Example to illustrate passing a pointer as a parameter.

//-------   Function f() ----------
void f( int * ip )
{
   *ip = *ip + 10 ;   //  Add 10 to the contents of memory.
}

//-------   Main Program ----------
int main()
{
   int m    ;   // Declare m to be an ordinary integer.
   int * p  ;   // Declare p to be a pointer to an integer.

   m = 12   ;   // Initialize m to be some number.

   p = &m    ;  // Initialize a pointer p to the memory location of m.

   // Print the number m.
   cout << "Before the call to f():  m = " << m << endl ;

   f(p) ;   //  Pass a pointer to the function f.   Function f then
            //  has the ability to change the contents of the memory
            //  indicated by the pointer.

   // Print the number m again.
   cout << "After the call to f():  m = " << m << endl ;
}
```

```
----------------------------- Sample Session -----------------------------
atlas% g++ param.cc
atlas% a.out
Before the call to f():  m = 12
After the call to f():  m = 22
```

6. Pointer coercion In this example, we look at an integer array of length two. Two very carefully chosen numbers are put into that array. The array is then coerced as a pointer to a character, and treated as if it were a character string.

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

//  Example to illustrate pointer coercion.


//-------   Main Program ----------
int main()
{
   int a[2]   ;  // Declare a to be a fixed-length array of length two.
   int  * ip  ;  // Declare ip to be a pointer to an integer.
   char * cp  ;  // Declare cp to be a pointer to an integer.

   a[0] = 1819043144  ;   // Carefully chosen numbers.
   a[1] = 111         ;

   ip = a    ;  // Initialize a pointer p to the memory location of m.

   cp = (char *) ip ;   // Pretend the pointer to an integer is really
                        // a pointer to a character.

   cout << cp << endl ;  // Print those numbers as if they were a string.
}


------------------------ Sample Session -----------------------
atlas% g++ coerce.cc
atlas% a.out
Hello
```

"Dirty" tricks with pointers

Pointers can be used to reveal details underlying the software architecture of a language system. Normally, it is not good programming style to use these tricks in well-written programs. But, these tricks can confirm our understanding of memory allocation.

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

//  Example to illustrate dirty tricks with pointers.
//
int main()
{
   int a, b, c  ;  // Declare three ordinary integers.
   int  * ip  ;    // Declare ip to be a pointer to an integer.

   a = 5 ; b = 7 ; c = 11 ;  // Initialize variables.

   // a, b, and c are allocated within the activation record for
   // the current execution context.   Many compilers will allocate
   // local variables in consecutive memory locations.
   // Subsequent variables are allocated in the direction of
   // numerically lower addresses.
   ip = &a ;

   cout << "*ip = " << *ip << endl ;
   cout << "*(ip-1) = " << *(ip-1) << endl ;
   cout << "*(ip-2) = " << *(ip-2) << endl ;

   //  Recall that integers occupy four bytes.  When we subtract one
   //  from the pointer ip, the compiler automatically converts "one" to
   //  correspond to "one integer", or 4 bytes.
}
```

```
------------------------- Sample Session -------------------------
atlas% g++ dirty.cc
atlas% a.out
*ip = 5
*(ip-1) = 7
*(ip-2) = 11
```

Why do we care ?

1. Pointers enable us to handle dynamically allocated arrays using multiple dimensions, e.g. 1-D, 2-D, 3-D, etc.
2. Pointers enable us to implement efficient data structures such as linked lists, trees, stacks, queues, heaps, and graphs.