Lecture Notes for Lecture 6 of CS 5001 (Foundations of CS) for the Fall, 2018 session at the Northeastern University Silicon Valley Campus.

Pointers and Pointer Function Parameters

Philip Gust,
Clinical Instructor
Department of Computer Science

Lecture 5 Review

- An *array* is a fixed-size, sequential collection of *elements* of the same type that occupy contiguous memory locations.
- Arrays have one or more *dimensions*; an element is *indexed* using a non-negative integer along each dimension.
- An array declaration includes element type, identifier name, and number of elements in each dimension.
- Left-hand value of indexed element is the location to store a value; right-hand value is the value at that location
- Indexed loops can be used to access elements of array; nested loops for multi-dimensional array.
- *Strings* are arrays of *char* with *'\0'* character after the last actual character; string length does not include *'\0'* character.

Pointer

Address-of Operator &

- Every variable or constant in C occupies a memory location when the program is in memory, and every location has a address.
- The address of any variable can be gotten using the *address* prefix operator '&'.

```
int num= 3;
printf("The address of 'num' is %p\n", &num); // pointer format spec
```

- Output: "*The address of 'num' is 0x7fff5e9a7bc8*"
- The C type of a memory address returned by the '&' operator is a *pointer* to the type of the variable or constant
- The printf '%p' format specifier is used to print a pointer.

Pointer

Pointer Type

- The pointer value can be stored in a variable that is declared as a pointer to a specific type.

  type **var-name*;

- Examples

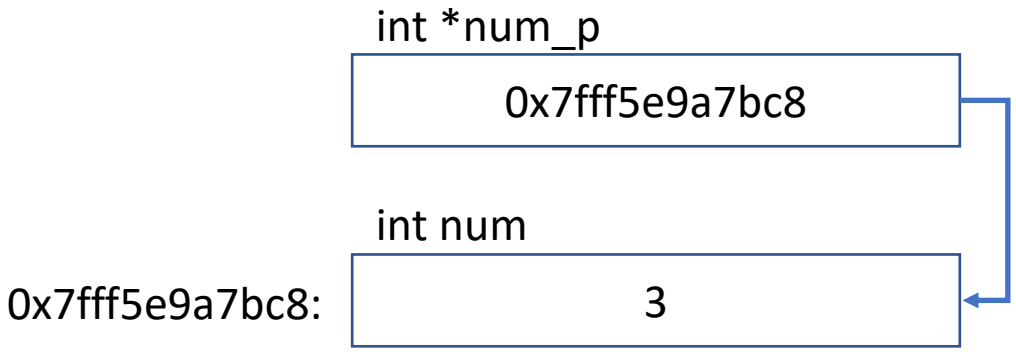
```
bool *bp; // pointer to a boolean
char *ch;  // pointer to a character
short *sp; // pointer to a short
int *ip; // pointer to an integer
long *lp; // pointer to a long
long long *lp; // pointer to a long
float *fp;   // pointer to a float
double *dp;  // pointer to a double
long double *ldp;  // pointer to a long double
```

Pointer

Pointer Type

- Here is the earlier code, this time storing the pointer before printing it:

```
int num= 3;
int *num_p= &num;  // type of num_p is “pointer to int”
printf(“The address of ‘num’ is %p\n”, num_p);
```



Pointer

Value Operator

- The value that is pointed to by a pointer variable can be gotten using the *value-at* operator '*'.

```
int num= 3;
printf("The value of 'num' is %d\n", num);  // print value

int *num_p= &num;  // type of num_p is "pointer to int"
printf("The address of 'num' is %p\n", num_p);  // print pointer

*num_p = 7;                                    // modify num through num_p
printf("The value of 'num' is %d\n", num);  // print value of num
```

- Output

The value of 'num' is 3
The address of 'num' is 0x7fff5e9a7bc8
The value of 'num' is 7

Pointer

Value Operator

- A pointer to a const must also be const to ensure that the const cannot be modified through pointer

```
const int num= 3;
printf("The value of 'num' is %d\n", num);  // print value

// int *num_p= &num;  // compiler warns about non-const pointer to const
const int *num_p= &num;  // ensures num cannot be modified through pointer

// num = 7;              // compilation error for const num
// *num_p = 7;           // also compilation error for const num_p to const num
```

Pointer

Void and Pointers

- We have seen the special type *void* several times.
  - Return type of a function that returns no value
  - As parameter list for function that has no parameters
- Can also be used as type of a pointer to anything; useful for functions that act on any pointer type

```
int num= 3;
void *voidp= &num;  // type of voidp is “pointer to anything”
```

- Void pointer must be assigned or cast to pointer to proper type before applying ‘*’ operator

```
int *intp = voidp;       // assignment “casts” null* to int*
printf("The value of ‘num’ is %d\n", *intp);  // print int at pointer
```

Pointer

NULL Pointer

- A variable of type pointer can be set to a special value indicating that the variable does not currently point to anything.
- The special constant *NULL* is defined as the address 0. Think of NULL as a void *pointer.
- The value-of operator (’*’) cannot be applied to the value NULL because memory address 0 is protected on most systems.

```
int *nullp = NULL;
printf("The address of ‘nullp’ is %p\n", nullp);
printf("The value of ‘nullp’ is %d\n", *nullp);  // access violation
```

Pointer

NULL Pointer and Programming Style

- Programs can test for NULL and take different steps in that case.
- Original Unix code treated pointers as a boolean values to test for NULL:

```
int *valp; …
if ( !valp ) {  // handle NULL pointer
}
```

- Developers of Unix discovered too late that this is hard to understand, and replaced it in later versions of Unix with

```
int *valp; …
If ( valp == NULL ) {  // handle NULL pointer
}
```

- Learn from their mistake: always use the latter form, never the former.

Pointer

What Are Pointers Used For?

- Pointers in C are used for several purposes:
  - Parameter passing
  - Passing functions to other functions
  - Alternative way to access arrays
  - Accessing interior fields of structures
  - Working with dynamic storage
- In this lecture, we will focus on parameter passing, and come back to the others in later lecture.

Passing Pointers as Parameters

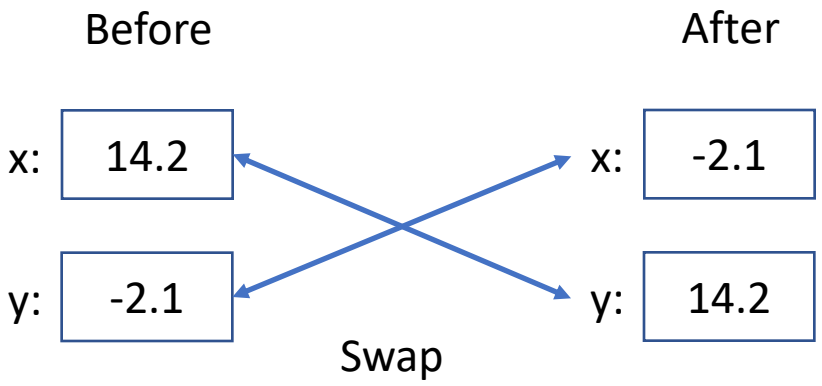
Why Pass Pointers as Parameters?

- C passes parameters to functions “by value”
  - Actual parameter values copied to formal parameter variables
  - Function treats formal parameters as local variables
  - Result copied to result variable
- Changes made to formal parameter values within function have no effect on the caller
- Usually that is a good thing, but not always.

Passing Pointers as Parameters

Example: Swap two values

- A common operation in data structures and algorithms like sorting is to swap the value of two variables.



- Preferable to create function that swaps values of two variables, but how with call by value?

Passing Pointers as Parameters

Example: Swap two values

- Implement “call by reference” using pointers to variables rather than variables themselves (pass pointers by value)

```
#include <stdio.h>

/**
 * Swap value of two variables.
 * @param val1p pointer to first variable
 * @param val2p pointer to second variable
*/
void swap( double *val1p, double *val2p) {
      int tmp = *val1p;  // save value of first variable
      *val1p = *val2p;   // copy second variable value to first
      *val2p = tmp;      // copy saved value to second variable
}
```

Passing Pointers as Parameters

Example: Swap two values

- Implement “call by reference” using pointers to variables rather than variables themselves (pass pointers by value)

```
/**
 * Test swap function for two variables.
*/
int main(void) {
	double firstVal = 14.2, secondVal = -2.1;
	printf("before: firstVal: %.1f, secondVal: %.1f\n", firstVal, secondVal);
	swap( &firstVal, &secondVal);
	printf("after: firstVal: %.1f, secondVal: %.1f\n", firstVal, secondVal);
}
```

Output:

```
before: firstVal: 14.2, secondVal: -2.1
after: firstVal: -2.1, secondVal: 14.2
```

Passing Pointers as Parameters

Example: Pointers to parameters to offset

- Pass pointers to points to offset

```
#include <stdio.h>
/**
 * Offsets a point x and y by an offset offX and offY
 * @param xp pointer to point x
 * @param yp pointer to point y
 * @param offX x offset
 * @param offY y offset
*/
void offsetPoint( double *xp, double *yp, double offX, double offY) {
        *xp += offX;
        *yp += offY;
}
```

Passing Pointers into Functions

Example: Pointer parameters to offset

- Pass pointers to points to offset.

```
/**
 * Test offsetPoint() function.
 * @return EXIT_SUCCESS
*/
int main(void) {
	double x= 14.2, y = -2.1;
	printf("Before x: %.1f, y: %.1f\n", x, y);
	offsetPoint(&x, &y, 10.0, 20.0);
	printf("After x: %.1f, y: %.1f\n", x, y);
}
```

Output:

```
After x: 14.2, y: -2.1
After x: 24.2, y: 17.9
```

Passing Pointers as Parameters

Example: Pointer to variable with greater value

- Pass pointers to variables; return pointer to variable whose value is greater.

```
#include <stdio.h>
/**
 * Returns pointer to variable whose value is greater.
 * @param val1 pointer to first variable
 * @param val2 pointer to second variable
 * @return pointer to variable whose value is greater
*/
double *maxp( double *val1, double *val2) {
      return (*val1 >= *val2) ? val1 : val2;
}
```

Passing Pointers as Parameters

Example: Pointer to variable with maximum value

- Pass pointers to variables; return pointer to variable whose value is greater.

```
/**
 * Test swap function for two variables.
 * @return EXIT_SUCCESS
*/
int main(void) {
	double firstVal = 14.2, secondVal = -2.1;
	double *mx = maxp(&firstVal, &secondVal);
	printf("maxp: %p, max value*maxp: %.1f\n", mx, *mx);;
	return EXIT_SUCCESS;
}
```

Output:

maxp: 0x7ffee76290c8, max value *maxp: 14.2

Passing Pointers as Parameters

Example: Pointer to variable with greatest value

- Pass null-terminated array of pointers to variables; return pointer to variable whose value is greatest.

```
#include <stdio.h>
/**
  * Returns pointer to variable in null-terminated
  * pointer array whose value is greatest.
  * @param vals array of pointers to values
  * @return pointer to variable whose value is
  *   greatest or NULL if list is empty
  */
double *maxpv( double *vals[]) {
       // if list empty, return NULL pointer
       if (*vals[0] == NULL) {
              return NULL;
       }
```

Passing Pointers as Parameters

Example: Pointer to variable with greatest value

- Pass null-terminated array of pointers to variables; return pointer to variable whose value is greatest.

```
        // use first entry as starting value
        double *maxp = vals[0];

        // find larger entry in list
        for (int i = 1; vals[i] != NULL; i++) {
            if (*maxp < *vals[i]) {
                maxp = vals[i];
            }
        }
        return maxp;
}
```

Passing Pointers as Parameters

Example: Pointer to variable with greatest value

- Pass null-terminated array of pointers to variables; return pointer to variable whose value is greatest.

```
/**
 * Test maxpv.
 */
int main(void) {
        double v1 = 14.2, v2 = -2.1, v3 = -10.9, v4 = 19.0, v5 = 5.1;
        double *values[] = { &v1, &v2, &v3, &v4, &v5, NULL };  // null-terminated array
        double *mx = maxpv(values);
        printf("maxpv: %p, max value *maxpv: %.1f\n", mx, *mx);
}
```

Output:

maxp: 0x7ffee7629090, max value *maxpv: 19.0