CS3157: Advanced Programming

Lecture #7

Oct 10

Shlomo Hershkop
shlomo@cs.columbia.edu

1

Outline

- Continuing with C
  - Control flow
  - Arrays
  - Pointers
  - strings
  - string library
  - string tokenizing
  - Memory allocation intro
- Reading
  - (k&r ch 4.1-4.3,7.1-7.5)

2

Announcements

- Midterm next Monday
- No Class Wednesday, use it to study
  - Will hold extra TA hours to help you with midterm study
  - Sample questions posted
  - Study homework assignment, notes, and labs

3

C control flow

- blocks are enclosed in curly brackets
- functions are blocks
- main() is a function
- blocks have two parts:
  - variable declaration (“data segment”)
  - code segment
- in C, variables have to be declared before they are used
- initializations can occur at the end of the declaration section, but before the code section

4

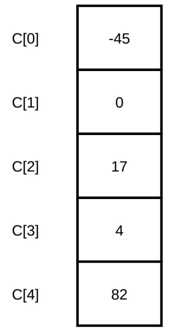
Intro arrays

- An array is a group of memory locations with the same name and type
- To get to a particular element in the array we need
  - data type
  - name
  - Length or position
- Array length can be determined:
  - statically— at compile time (when we code)
    - e.g., char str1[10];
  - dynamically— at run time (more on this later)
    - e.g., char *str2;

- defining a variable is called “allocating memory” to store that variable
- defining an array means allocating memory for a group of bytes,
- individual array elements are indexed
  - starting with 0
  - ending with length -1
- indices follow array name, enclosed in square brackets ([ ])

  e.g., name[25]

int C[5]



We can say for example

printf (“ %d”, C[2]);

And

X = C[4] / 5;

Declarations:

int b[100],v[3];

int array

```
#include <stdio.h>
#define MAX 6

int main( void ) {
int arr[MAX] = { -45, 6, 0, 72, 1543, 62 };
int i;

for ( i=0; i<MAX; i++ ) {
    printf( “%d = %d", i, arr[i] );
}

printf( "\n" );
} /* end of main() */
```

More arrays

- Can also create arrays in the following manners

1. int a[] = {1,2,3};
2. int b[3] = {6,3,7};
3. int n[10] = {0};

Note you need to initialize the array elements, 3 is a trick case.

9

Pointer power

- Variables that contain memory addresses as their values
- Data types we've learned about in C use direct addressing
- Pointers facilitate indirect addressing
- Declaring pointers:
  – pointers indirectly address memory where data of the types we've already discussed is stored (e.g., int, char, float, etc.)
  – declaration uses asterisks (*) to indicate a pointer to a memory location storing a particular data type
  – Called dereferencing a pointer
- example:

```
int *count;
float *avg;
```

10

Pointers: nitty gritty

- ampersand & is used to get the address of a variable

- example:

int count = 12;

int *countPtr = &count;

- &count returns the address of count and stores it in the pointer variable countPtr

11

Another example

- here's another example:

```
int i = 3, j = -99;
int count = 12;
int *countPtr = &count;
printf ( “%d”, *countPtr);
```

- Here is the memory picture:

12

Arrays as pointers

- an array is some number of contiguous memory locations
- an array definition is really a pointer to the starting memory location of the array
- and pointers are really integers
- so you can perform integer arithmetic on them
- e.g., +1 increments a pointer, -1 decrements
- you can use this to move from one array element to another

13

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
int i, *j, arr[5];
srand( time ( NULL ));
for ( i=0; i<5; i++ )
arr[i] = rand() % 100;
printf( "arr=%p\n",arr );
for ( i=0; i<5; i++ ) {
printf( "i=%d arr[i]=%d &arr[i]=%p\n",i,arr[i],&arr[i] );
}
j = &arr[0];
printf( "\nj=%p *j=%d\n",j,*j );
j++;
printf( "after adding 1 to j:\n j=%p *j=%d\n",j,*j );
}
```

14

Output

```
arr=0xbffff4f0
i=0 arr[i]=29 &arr[i]=0xbffff4f0
i=1 arr[i]=8 &arr[i]=0xbffff4f4
i=2 arr[i]=18 &arr[i]=0xbffff4f8
i=3 arr[i]=95 &arr[i]=0xbffff4fc
i=4 arr[i]=48 &arr[i]=0xbffff500
j=0xbffff4f0 *j=29
after adding 1 to j:
j=0xbffff4f4 *j=8
```

15

Pointer operations

- Difference between
  - ptr++
  - *ptr++
- int b[5] ….
  int *bPtr;

  bPtr = b   //or
  bPtr = &b[0]

16

- Careful when moving pointers
- bPTr += 2;

  the memory location isn't simply incremented by 2…..depends on size of type being pointed to.

17

Strings

- storing multiple characters in a single variable
- data type is still char
- BUT it has a length
- last character the is terminator: '\0', aka NULL
- string constants are surrounded by double quotes: "
- example:
  - char s[6] = "ABCDE";

18

String II

- char s[6] = "ABCDE";
- Memory storage looks like:

A	B	C	D	E	\0

- Need to remember that you are really accessing indices 0 – (*length*-2) since the value at *length*-1 is always \0

19

Using strings

- printing strings
- format sequence: %s
- example:

```
#include <stdio.h>
int main() {
char str[6] = "ABCDE";
printf( "str = %s\n", str );
} /* end of main() */
```

20

String Library

- to use the string library, include the header in your C source file:

```
#include <string.h>
```

- string length function:

```
int strlen( char *s );
```

- this function returns the number of characters in s; note that this is NOT the same thing as the number of characters allocated for the string array

- string comparison function:

```
int strcmp( const char *s1, const char *s2 );
```

- “This function returns an integer greater than, equal to, or less than 0, if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2 respectively. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes that differ in the strings being compared.”

man strcmp

21

copying functions:

```
char *strcpy( char *dest, char
  *source );
```

- copies characters from source array into dest array up to NULL

```
char *strncpy( char *dest, char
  *source, int num );
```

- copies characters from source array into dest array; stops after num characters (if no NULL before that); appends NUL

22

Search

```
char *strchr( const char
  *source, const char ch );
```

- returns pointer to first occurrence of ch in source; NULL if none

```
char *strstr( const char
  *source, const char *search );
```

- return pointer to first occurrence of search in source

23

String Parsing

```
char *strtok( char *s1, const char
  *s2 );
```

- breaks string s1 into a series of tokens, delimited by s2
- called the first time with s1 equal to the string you want to break up
- called subsequent times with NULL as the first argument
- each time is called, it returns the next token on the string
- returns null when no more tokens remain

24

Example

```
char inputline[1024];
char *name, *rank, *serial_num;
printf( "enter name+rank+serial number: " );
scanf( "%s", inputline );
name = strtok( inputline,"+" );
rank = strtok( null,"+" );
serial_num = strtok( null,"+" );
```

25

Formatting functions

`int sscanf(char *string, char *format, ...)`

- parse the contents of string according to format
- placed the parsed items into 3rd, 4th, 5th, ... argument
- return the number of successful conversions

`int sprintf(char *buffer, char *format, ...)`

- produce a string formatted according to format
- place this string into the buffer
- the 3rd, 4th, 5th, ... arguments are formatted
- return number of successful conversions

- format characters are like printf and scanf (see notes from earlier lectures)

26

Memory allocations

- One of the most powerful features of c is the ability of the programmer to create more memory space during the execution of the program.
- Limited by physical machine memory
- If you want to be able to create memory, you also need to free it manually

27

malloc /sizeof / free

- charPtr = malloc (sizeof (…));
- free (charPtr)

28

Review for midterm

- Will post online review notes, and some sample questions etc.
- Open book, open notes
- No computers or any computing device
- Will be testing you on understanding of material covered

29

- Should know perl
  - Basic types and usages, comments etc.
  - Perl debugger
  - File handling, Subroutines
  - md5
- CGI
  - What, how, and sometimes why
  - How to use perl/CGI
- C
  - Basics
  - Types/pointers/arrays
  - Linking, compiling, and makefiles

30