Lecture 6: Arrays, Structs, Strings, Pre-Processor

Tyler Hendrick, Matthew Heilman

Arrays

Definition

- An array is a collection of consecutive elements in the stack of the same data type.

Declaration

- There are two methods of declaring an array
    1. data_type variable_name[size];
        - Used to declare an array of a known size
        - The size in the square brackets must be a constant value, not a variable
        - Memory allocated in the stack
    2. data_type * variable_name = malloc(quantity*sizeof(data_type);
        - Used to declare an array of dynamic size
        - Memory allocated in the heap

```
void declareArrayExample(){
        const int size = 5;
        int length = 5;
// methods that work
        int a[8];
        int b[size];
        int d* = malloc(8 * sizeof(int) );
        int e* = malloc(length * sizeof(int)) ;          // this works because size is a constant

// method that do not work
        int c[length];                    // this does not work because the value of length varies
}
```

Accessing element values in an existing array

- There are two methods of accessing array elements
    1. variable_name[index]
        - index goes from zero to (size-1)
    2. *(variable_name + index)
        - pointer arithmetic
        - here, variable_name is a pointer to the starting address of the array and the index is number of how many elements after the start you want to access

```
void accessArrayExample(){
	int a[3];
	int *b = malloc(3*sizeof(int) );
	a[0] = 1;
	a[1] = a[1] + 2;		// equivalent to a[1] = 1 + 2 = 3

	*(b) = 1;
	*(b+1) = *b + 2;			// equivalent to *(b+1) = 1 + 2 = 3

	a[2] = *(b) + a[1];			// equivalent to a[2] = 1 + 3 = 4
	*(b+2) = a[2] + *(b+1)		// equivalent to *(b+2) = 4 + 3 = 7
}
```

Multi-dimensional Arrays

- an object which has a length and a width characteristic
- works like a one-dimensional in regards to creation and accessing elements

```
void multiArrayExample(){
	int a[2][3];			// int *a = malloc(2 * 3 * sizeof(int)
	a[0][0] = 1;			// *(a + 0*3 + 0*2) = 1
	a[1][2] = 6;			// *(a + 1*3 + 2*2) = 6
}
```

Strings

Definition

- Strings are arrays of data type character (char)

Useage

- Strings can be declared by using either:
    - char*
    - char[]
        - ex: char str[]
        - char *str
- The null character “\0” ends every string
    - char str[] = {‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’} is the same as char str[] = “Hello”
- The printf format is:
    - %s
- (str + 5) will give you a substring that starts at the 6th letter of the string
    - ex: char str[] = “Winchester”;
      printf(“The substring is %s”, (str + 5));
      This would print out “The substring is ester”
- *(str + 5) will give you only the 6th character
    - Using the example above, but using *(str + 5) instead would have given you “The substring is e”

Functions

- The preprocessor file, string.h, has many prebuilt functions that make it easier to work with strings.
- Examples:
    - strlen(s)
        - Returns the length of the string, not including ‘\0’
    - strncpy(dst, src, n)
        - Copies n characters from the source, src, and adds them onto destination, dst, including the ‘\0’
    - strncat(dst, src, n)
        - Copies characters from the source, src, and adds them onto destination, dst, until the destination has n characters, including ‘\0’
    - int strcmp(char *str1, char *str2)
        - Compares the two strings.

        - Returns a 0 if the strings are equal, a positive number if the first string is greater than the second, or a negative number if the second string is greater than the first using ASCII order.
    - char *strstr(char *str1, char *str2)
        - Searches for a substring, str2, in str1.
    - char *strdup (char * str)
        - Dynamically allocates space on the heap and copies the argument, str, into the space
            - The allocated space must be freed when done
    - char *strtok_r(char *str, char *delim, char **sav)
        - Breaks a string into pieces

Complex Data Types

C has different data types that are more complex than the standard ones.

Enumeration

- An Enumeration is a set of named integer constants
- Declaration:

  ```
  enum identifier {enumerator-list};
  ```
- example

  ```
  enum day {Sunday = 0, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
  ```

Structures

- A Structure is a user created data type that is a combination of other data types
- Declaration:

  ```
  struct identifier {data types};
  ```
- example

  ```
  struct mystruct { char name[32];
          int age;
          char *addr;
  };
  ```

Union

- A Union allows multiple data types to be stored in the same memory location
- Declaration:

  ```
  union identifier {member definitions};
  ```
- example

  ```
  union data{int i;
          float f;
          char c;
  };
  ```

Function Pointers

- A Function Pointer stores the address of a function to later be called.
- Declaration:

  ```
  DataType (*pointer_name)(function arguments);
  ```
- example

  ```
  int (compare_cb)(int a, int b);
  ```

The Preprocessor

#include

- will insert code from specified files before compiling the project

#define

- creates a macro
- replaces the first argument with all that follows

#ifdef/ifndef #endif

- used in conjunction to #define to prevent duplicate copies of code