Lab Manual Section 1: The C Programming Language

1-2: Pointers

Objectives

1. Introduce students to pointers in C including the relationship between array indexing and pointer arithmetic.
2. Have the ability to use pointer to pass the address of data to functions

Further Reading

1. The C Programming Language, page 93

Introduction to Pointers

Pointers in C are simply variables that hold a memory address. Take a look at this fragment of sample code (with line numbers added):

```
1:  char *string = "This is a string.";
2:  char *index;
3:
4:  index = string;
5:
6:  printHex(index);
7:  putchar(*index);
```

Let's analyze what's going on here, line by line:

- **Line 1.** We are simply defining a string. In C, a string is simply an array of characters. As you will learn later, the concepts of arrays and pointers in C are very closely related: this is why we are able to define a string in this way. This line of code is essentially saying "let the variable `string` be a pointer of type `char` to the string 'This is a string.'."
- **Line 2.** We are declaring a new pointer variable of type `char` named `index`.
- **Line 4.** We are letting the value of `index` be the value of `string`. More simply put, we are letting `index` point to the same thing as `string`.
- **Line 6.** We print the value of `index` in hexadecimal. Since `index` is a pointer, this will print the *address in memory* pointed to by `index`.
- **Line 7.** We print the value of *the character pointed to* by `index`. This is known as *dereferencing* the pointer.

```
    char *string = "This is a string.";
    char *index;

    index = string;

    printHex(index);
    putchar('\n');
    putchar(*index);
```

```
    putchar('\n');

    index++;

    printHex(index);
    putchar('\n');
    putchar(*index);
    putchar('\n');
```

Now, create a new charPointer.c file in this lab's exercise directory. Copy and paste the fragment of code above into that file charPointer.c. Create a make file and include your charPointer.c and printHex.c (from the previous lab) file to compile then run the program. Create a text file `charPointer.txt` and write down your output and explain how the address and the content of index pointer changes. Make sure that you add and commit both files to the repository.

```
int x[10];

int *y;

y = &(x[0]);
printHex(y);
putchar('\n');

y = &x;
printHex(y);
putchar('\n');

y++;
printHex(y);
putchar('\n');
```

Note that using the `&` symbol (as on lines 5 of the sample code) means "the address of".

Create another intPointer.c file in this lab's exercise directory. Copy and paste the fragment of code above into that file (intPointer.c). Create a make file and include your intPointer.c and printHex.c (from the previous lab) file to compile then run the program. Create a text file `intPointer.txt` and write down your output and explain how the address of y pointer changes and why it differs from the char pointer. Make sure that you add and commit both files to the repository.

Assignment

Take a look at `print.c` in your lab exercise directory. You will notice that it contains a function named `print_string()`. Your assignment is to modify this program to use a character pointer to traverse and print the string instead of array indexes. This means that your finished code should no longer use the `i` variable.

Hint: In C, strings are automatically terminated with a zero character—that is, the last character in any C string has the numeric value 0. This is called the *null terminator*; this is the character called NUL in many ASCII tables. How can you use the fact that every string in C is terminated with a null character to determine when to stop printing characters?

1-1: Shifts and Masks **1-2: Pointers** 1-3: Signed and Unsigned Numbers