Lab 4: Pointers

Objectives

The main goal of this laboratory is to:

- Understand how variables of different types are stored in memory and use pointers to access their addresses.
- Apply pointers to work with arrays, strings, and functions.
- Implement and test basic string functions using pointers, demonstrating how strings are manipulated without standard library functions.

Basic Pointer Operations in C

Review the provided notes before starting the exercises.

Understanding Pointers

1. Write a short C program (`lab4ex1.c`) that declares and initializes (to any value you like) a `char`, an `int`, a `double`, and a `long`. Next declare and initialize a pointer to each of the four variables. Your program should then print the address of each variable and the number of bytes each occupies in memory. Use the `%p` formatting specifier to print the address in hexadecimal. Use the `sizeof` operator to determine the memory size allocated for each variable. Your output should look like:

   ```
   char: size = sizeof char, address = address of variable
   int: size = sizeof int, address = address of variable
   double: size = sizeof double, address = address of variable
   long: size = sizeof long, address = address of variable
   ```

   Compile your program with:

   ```
   $ gcc -o lab4ex1 lab4ex1.c
   ```

   Run your program with:

   ```
   $ ./lab4ex1
   ```

   Now compile your program with:

```
$ gcc -m32 -o lab4ex1 lab4ex1.c
```

where the `-m32` flag indicates that the compiler will generate code for a 32-bit machine. Observe the difference between the two versions.

STOP RIGHT HERE!

2. In the following C program (`lab4ex2.c`), find out (add code to print) the address of the variable `x` in `func1`, and the variable `y` in `func2`. What did you observe? Can you explain this?

```
#include <stdio.h>

void func1(int xval)
{
   int x;
   x = xval;
   /* print the address and value of x here */
}

void func2(int dummy)
{
   int y;
   /* print the address and value of y here */
}

void main()
{
   func1(7);
   func2(11);
}
```

STOP RIGHT HERE!

3. The program below (`lab4ex3.c`) uses pointer arithmetic to determine the size of a `char` variable. By using pointer arithmetic we can find out the value of `cp` and the value of `cp+1`. Since `cp` is a pointer, this addition involves pointer arithmetic: adding one to a pointer makes the pointer point to the next element of the same type. For a pointer to a `char`, adding 1 really just means adding 1 to the address, but this is only because each `char` is 1 byte.
   A. Compile and run the program and see what it does.
   B. Write some code that does pointer arithmetic with a pointer to an `int` and determine how big an `int` is.
   C. Same idea – figure out how big a `double` is, by using pointer arithmetic and printing out the value of the pointer before and after adding 1.
   D. What should happen if you added 2 to the pointers from part (a) through (c), instead of 1? Use your program to verify your answer.

```
#include <stdio.h>

void main( )
{
   char c = 'Z';
```

```
        char *cp = &c;

        printf("cp is %p\n", cp);
        printf("The character at cp is %c\n", *cp);

        /* Pointer arithmetic - see what cp+1 is */
        cp = cp + 1;
        printf("cp is now %p\n", cp);

        /* Note: Do not try to print *cp here, because it points to the
           memory location unallocated to your program */
     }
```

STOP RIGHT HERE!

Pointers and Strings

1. **Computing the length of a string**: Using pointers write a C function:

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

   which returns the length of `s`, that is, the number of characters between the beginning of the string and the terminating null character.

2. **Copying strings**: Using pointers write a C function:

   ```
   char *mystrcpy(char *dest, char *src);
   ```

   which copies characters from `src` to `dest` strings and returns `dest`. Make sure `dest` is null-terminated.

3. **Concatenating strings**: Using pointers write a C function:

   ```
   char *mystrcat(char *dest, char *src);
   ```

   which appends characters from `src` to the end of `dest` and returns `dest`. Make sure `dest` is null-terminated.

Use the following driver (`mystr.c`) to test your functions:

```
#include <stdio.h>

void main()
{
   char *s1 = "Hello ";
   char *s2 = "CPSC 275!";
   char s3[20];

   printf("length of s1 = %d\n", mystrlen(s1));
   printf("s3 = %s\n", mystrcpy(s3, s1));
   printf("s3 = %s\n", mystrcat(s3, s2));
}
```

Handin

When completed, ask your instructor or TA to check your work.

- **Welcome: Sean**
  - LogOut