COMP 235 Lab #2: Command-line args and malloc/free

The objective of this lab is for you to practice reading command-line arguments and to use malloc/free for dynamically allocating strings.

Step 1: Reading command-line arguments

Use ssh to login to deepblue. Create a new directory for this lab. Using any text editor (pico, vi, emacs, …) that you are familiar with, create a file called main.c that contains the following.

```
#include <stdio.h>

int main(int argc, char** argv)
{
   for (int i=0; i < argc; i++)
     printf("%s\n", argv[i]);
}
```

The function declaration for main is used to read command-line arguments passed to the program when invoked from the OS prompt.

`char** argv` is equivalent to `char* argv[]`. In other words, `argv` is a variable-length array of strings. The parameter `argc` is used to tell us how many strings it contains.

Compile and run your program, passing it any arguments you like. For example, if you invoke your program as follows:

```
./a.out foo 1 fee "this is an arg" onemore
```

…then the output would be:

```
./a.out
foo
1
fee
this is an arg
onemore
```

Note that argv[0] is always the name of the process.

Step 2: Memory allocation with malloc/free

Check out the following program:

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

void printEm(int *a, int length)
{
     int  x;

     for (x = 0; x < length; x++)
        printf("%d ",a[x]);
     printf("\n");
}

int main()
{
     int  x, *arrayOfInts, length;
     char y, *s, str[20];

        // do some stuff with pointers...
     s = &y;

     *s = 'D';

     strcpy(str, "Hello, World.");

        // now print them
     printf("%c %c %s\n",*s,y,str);

        // allocate an array of integers
     printf("Enter a size for array of ints: ");
     scanf("%d", &length);
     arrayOfInts = (int *)malloc(sizeof(int) * length);

     for (x = 0; x < length; x++)
        arrayOfInts[x] = x*x;

     printEm(arrayOfInts, length);
     // equivalent to: printEm(&arrayOfInts[0], length);

     free(arrayOfInts);
}
```

Study this program to review the concepts of pointers and memory allocation with `malloc` and deallocation with `free`.

Change this program so that it reads in the size of the array as a command-line argument.

Step 3: String allocation

Starting with your main routine from Step 1, add two functions that meet the following specifications:

```
/* Returns a null-terminated reversed version of the string
passed as an argument into a newly allocated array of
characters. The calling function is responsible for
free'ing the memory allocated by this function. */
char* strrev(char* s);

/* Returns 1 if the string passed as an argument is a
palindrome. Returns 0 if the argument is not a palindrome.
Returns -1 if passed a NULL argument. */
int isPalindrome(char* s);
```

Using these functions, modify your main routine so that it creates a palindrome out of its command-line arguments and prints them to standard output one per line. If a command line argument is already a palindrome, print it out untouched. Call your program `pal`.

For example, if you called your program as follows:

```
pal foo 1 feef Fred "this is big"
```

…the output should be:

```
ooffoo
1
feef
derFFred
gib si sihtthis is big
```

Your program should check to ensure there is at least one command-line argument provided, and if not, should generate a usage hint like the following:

```
Usage: pal str [str]*
```

Step 4: Demonstration

In order to receive full marks for this lab, demonstrate your program to the instructor in the lab during the week it is due.