Arrays and Pointers

Lab 6

The labs, for this course, are designed to be completed on your own at home or in the 3rd floor Trottier labs. These labs are not graded. You do not hand in these labs. If you prefer to work on a lab in a supervised setting, check the TA Information schedule for the Lab TA period(s). You will find this schedule in our MyCourses page under Content/Course Information. The supervised labs are not teaching environments. The Lab TA will simply be present to answer questions and provide support.

This lab is about C string arrays and the relationship between arrays and pointers. The string library provides common utilities that are used to manipulate strings more efficiently. In the following we will try to understand how theses utility functions are implemented. The implementations provided are not complete solutions but they give an idea about what goes behind the scenes when you call one of the functions provided by string.h.

Part one: finding string length

strlen is a function provided by string.h that returns the number of characters (excluding the termination character \0) of the string provided as input.
For example:

```
char mystring[] = "Hello Comp206 class";
strlen(mystring); // returns 19
```

Now have a look at the following code:

```
unsigned int comp206_strlen(char* ptr)
{
      unsigned int length = 0;
      while(*ptr!='\0')
      {
        ++ptr;
        ++length;
      }
      return length;
}
```

Try to answer the following questions:

1. From a visual inspection, what does the code do exactly?
2. Why does it return unsigned int instead of int?
3. Make sure to make this code compile and run on mimi and feel free to experiment with it.
4. Note that the input parameter is of type char*. Can you still use comp206_strlen() function on the char array declared in the previous example?

```
char mystring[] = "Hello Comp206 class";
```

Part two: string comparison

strcmp is a function provided by string.h that compares two strings provided as input. It returns 0 if the strings are equal. Returns positive number if string1 is greater than string2, negative number if string2 is greater than string1.

Now have a look at the following code:

```
int com206_strcmp(char* array1, char* array2)
{
   while (*array1 == *array2) {
      if (*array1 == '\0' || *array2 == '\0')
      break;
      array1++;
     array2++;
   }

   if (*array1 == '\0' && *array2 == '\0')
      return 0;
   else
      return -1;
}
```

Try to answer the following questions:

1. From a visual inspection, what does the code do exactly?
2. Why does it return int instead of unsigned int?
3. Make sure to make this code compile and run on mimi and feel free to experiment with it.
4. Note that the return value is 0 if strings are equal and -1 otherwise. Can you modify the code to mimic the return value convention used by the strcmp function?
5. Rewrite the code using array notation instead of pointer notation

Part three: multidimensional arrays

When we initialize a one-dimensional array during declaration, we can omit to provide the size of the array because the compiler can figure it out from the size of the initialization list. For example:

```
int array[] = {1, 2, 3 ,4 };
```

is equivalent to:

```
int array[4] = {1, 2, 3 ,4 };
```

In the case of 2-dimensional arrays, things are a little bit more complicated. Check the following declarations:

```
int array1[2][2] = {1, 2, 3 ,4 };
int array2[][2] = {1, 2, 3 ,4 };
int array3[2][] = {1, 2, 3 ,4 };
int array4[][] = {1, 2, 3 ,4 };
```

Which of the previous declarations are invalid and why?

Now have a look at the following code:

```
int main()
{
   int array[3][2];
   int i =0;
   int j;
   for(; i<3; i++)
   {
      for(j=0;j<2;j++)
     {
        printf("Enter a value for element [%d][%d]:", i, j);
        scanf("%d", &array[i][j]);
      }
   }
   for (int i=0; i<=2; i++)
   {
      printf("%d ",array[i]);
   }
   return 0;
}
```

Try to answer the following questions:

1. From a visual inspection, what does the code do exactly?
2. Make sure to make this code compile and run on mimi and feel free to experiment with it.
3. Rewrite the code using pointer notation instead of array notation