Programming in C

Duration time: 90 minutes

Introduction

Requirements

- Pointers – part III.

Problem 1

Write a `swap()` function with the following function header:

```
void swap(int *p1, int *p2);
```

Use this function to swap the values of two integers.

```
int main(){
    int x = 10;
    int y = 20;

     // You must figure out how to call the function correctly!
    swap(...)

    // Should print out x: 20, y: 10
    printf("x: %d, y: %d\n", x, y);
}
```

Problem 2

Write a program that demonstrates the fact that arrays themselves are not passed to functions, but a pointer to the first element of the array is what is passed.

Hint: The `sizeof` operator will be useful.

Problem 3

According to the C standard, `arr[0]` is actually syntactic shorthand for `*(arr+0)`. Write a program that prints all elements of an integer array using this alternative notation.

Problem 4

Create a simple function `print_addr(int x)` whose sole purpose is to print the address of the integer `x` passed to it. Create an integer variable in main, print out its address, and then pass that variable to `print_addr`. Compare the results. Is this expected behavior?

Problem 5

Create a function `new_integer()` that declares and initializes an integer inside the function and returns the address of that integer. Print out the integer value associated with this memory address in main. Is this legal C? Does your complier display warnings? Is this a safe operation?

Problem 6

Create a function `print_array` which will print out all values of an integer array. What parameters must the function have in order to work for any integer array?

Problem 7

Write a program which uses an `ARR_SIZE` constant via `#define ARR_SIZE 10`. Create a function that allocates memory for `ARR_SIZE` integers, assigns the 0th integer to 0, the 1st integer to 1, and so on, and returns the address of the allocated space. (Your array should look like `[0|1|2|...]`). use the `print_array` function from problem 6 above to print out each element of the allocated array. After you have successfully made the program, alter the value of `ARR_SIZE` and observe the results.

Pointers are already confusing, but they could get even more complicated when they are used in conjunction with const. This section is intended to clear you of any doubt you may have regarding manipulating const pointers. Let's examine each case with examples.

Case 1: a const pointer pt that points to an int

```
int apple = 10;
int * const pt = &apple;
```

This declaration says that pt can point to only apple and nothing else. Therefore, you cannot make it point to another address later in the program. However, you can modify the value of apple through *pt or apple.

Case 2: a pointer pt that points to a const int

```
int apple = 10;
const int * pt = &apple;
```

This declaration states that pt points to a const int; so you cannot use pt to modify the value of apple. However, apple is not const, so you can change the value of apple by assigning a new value to apple. Also, pt can point to another address later.

Case 3: a pointer that points to an int

int apple = 10;
int * pt = &apple;

This declaration is what we normally use. You can change the value of apple through apple and *pt, and you can make pt point to another address later.

Case 4: a const pointer that points to an int, which is declared to be const

const int apple = 10;
int * const pt = &apple;

You should know from the first case that pt cannot point to another variable due to its const status. However, what about modifying the value of apple? If you think about it a little, you may figure out something incoherent is going on here.

If apple is made const, its value is not supposed to change no matter what. However, it seems as if you can use pt to change apple's value.