Homework 1 Report

1.

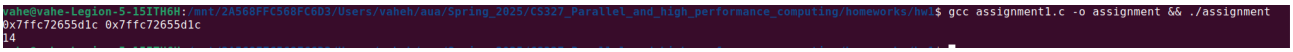
1. Declare int i = 5

2. create pointer p which points to the address of i

3. print the address using directly &i, and p

4. dereference p by doing *p, and add 9 to the underlying int

5. print again



2.

First I create print function, which will be used multiple times to print the value of the array, just so I don't duplicate the code. The print function

1. Takes as an argument a pointer to int (pointer to the first element of the array), and the size of the array.

2. Copies the value of the pointer to pointer p. Need this so that in the while loop I know when to stop

3. In a while loop, while p < a + size ... Basically we loop until p reaches the last element.

4. Dereference p and print the underlying int, also increment p. *(p++)

Then in the main

1. I initialize int array a of size 5 with values. int a[5] = ...

2. print a by using print function. a variable is itself a pointer to the start of the array, so we can just pass it to print like print(a)

3. then do a for loop for 5 iterations, in the loop dereference i-th element of a by doing *(a+i), and just assign new values to each element. In this case I just reverse the order.

4. print a again using print function

5. print a again just this time in a for loop, and printf side by side a's values by using a[i] notation, and also *(a+1) notation.

3.

First I create the swap function. It

1. Takes as arguments two integer pointers a, b

2. Creates a new int c and assigns it the dereferenced value of a by int c = *a;

2. Dereferences a, and assigns dereferenced value of b to it *a = *b;

3. Dereferences b and assigns the value of c to it.

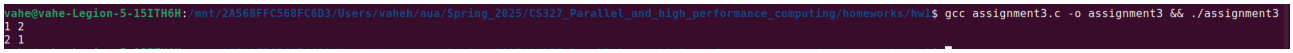
Then in main

1. create two ints a,b

2. print them

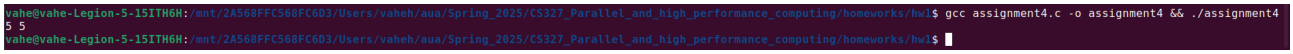
3. call the swap function, pass their addresses as arguments swap(&a, &b);

4. prints a, b again



4.

I'm not sure what to write about this one, because the code exactly does what the assignment says. Here's the output.



5.

1. create an int pointer a, and assign to it the result of malloc. The argument of malloc is sizeof(int), to allocate memory with size of a single int

2. dereference a, assign value to it. *a = 5

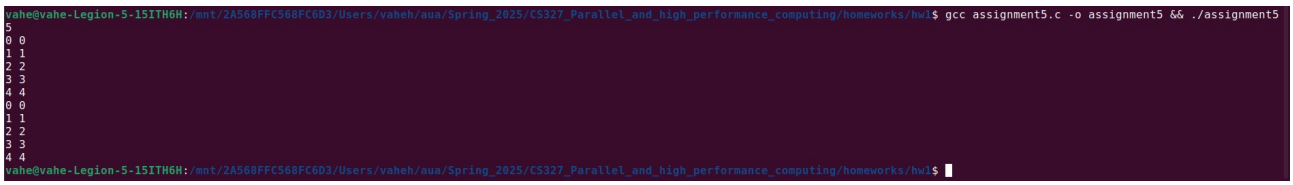
3. I used two notations to allocate memory for 5 element int arrays sizeof(int[5]), (sizeof(int) * 5). They have the same results. So I assign pointers to the start of two arrays with size 5 to int*b and int*c

4. Do a for loop with 5 iterations, in each iteration assign i to ith element of both b and c. I again used two different notations. One with pointer arithmetic *(c+i) = i; and one using i-th array element subscript operator b[i] = i;

5. In the loop print each element by using b[i], and dereferencing c by *c

6. Then call free on all a,b,c because they've all been dynamically allocated using malloc

7. Then I print the contents of b,c again to see what happens after free



6.

First I create a function str_length to calculate the length of a given string. It takes a char pointer.

1. declare unsigned int i = 0; this i will serve as our counter

2. using pointer arithmetic in the while loop condition, check if the current char is '\0' character, which marks the end of the string. while(*(str + (i++)) != '\0') {}

*(str + (i++) dereferences str's i-th char, and also increments i. The body of the loop does nothing since we're only interested in i's value

3. return i - 1, because the last iteration of the loop also incremented i

Then in main

1. create two char pointers, c, p. Assign a string to c, and assign c to p. p is just used for printing c using pointer arithmetic

2. do a while loop, while the dereferenced value of p != '\0' print the character, increment p.

3. print a new line

4. print the length of the string using the str_length function we created earlier

5. then print the string

please input a string: "

when you enter a string, I wanted to add the second double quote to the right of it, so it would show up like

please input a string: "string"

but when you press enter, the cursor goes to the next line. So I need to move the cursor back up, and all the way to the right of the line, so that the second " can be placed.

This is achieved by

6. printing "\033[A" escape sequence, which moves the cursor one line up. \033 is escape character, [is opening bracket, A is command to move up

7. Then I need a string that takes the cursor to the end of the line. I didn't find a command to just go all the way to the right, but I found how to move the cursor right as many columns as I need.

The problem here was that I didn't know how long the input string is going to be beforehand. I need a string like \033[10C which will move the cursor 10 columns to the right for example. I needed a

way to format the length of my line into this command. So I created a char pointer go_to_end which is going to hold the escape sequence of going right. I use sprintf to format the string I need and save it in go_to_end.

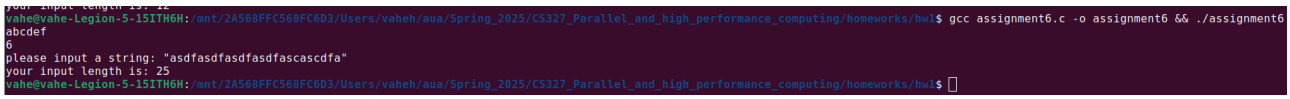
sprintf(go_to_end, "\033[%dC", 24 + str_length(user_input))

passes the go_to_end pointer to sprintf, which will save the string in it. The last argument is 24 + str_length(user_input) because 24 is the length of the string "please input a string: \"".

8. print the go_to_end escape sequence. I passed printf(go_to_end, 1); second argument to printf because I was getting warnings about printf being used without second argument

9. print " at the end of the sentence

10. go to new line and print the length of the user input string



7.

First I create a function to print string. It takes char pointer, prints the dereferenced value of the pointer, and increments it until '\0' is reached. Then prints a new line.

Then in main

1. create an array of size 10 of char pointers.

2. in a for loop, assign i-th element of that array the result of malloc call. I used different sizes for each malloc, just for experimenting. The result is cast to char* because I will be using these as strings.

3. in a nested for loop I assign values to each element. Just

for (; j < i + 1; ++j)

    arr[i][j] = '0' + j;

So take i-th element of the array of char pointers, take the j-th element of that string, assign to it '0' + j, which is just calculating the character of digit j. j is between 0 and 9.

4. At the very end of each string, set the value '\0'

5. print the string using print_string function

6. Then I used pointer arithmetic to modify the values, like

((arr + 2) + 1) = '9';

this dereferences 3rd element of arr with *(arr + 2), then because that's a char pointer, dereference the 2nd element of that by *(*(arr + 2) + 1), and just assign to it a new character, in this case '9';

And so on for a couple more strings

8. Then create a double pointer char** p = arr; because arr is an array of char pointers, that makes the arr itself a pointer to a character pointer, thus double pointer;

9. then do a while loop, dereference current string by doing *p, and pass it to the print_string function to print.

10. free current *p, because it's dynamically allocated. I guess I could have omitted this here because the program is going to end shortly, but I think it's best to make a habit of freeing memory and not forgetting about it.

11. increment p to go to the next char pointer.