

CS1010 Tutorial 5

Group BC1A

1 October 2020

Topics for today

Objectives

- Recap on Topics (Pointers, Array)
- Going through problem set 14, 15
- Common issues with Assignment 1 3.
- ~~Assignment 2~~
- Summary

Items that are graded

- *Assignment 3*

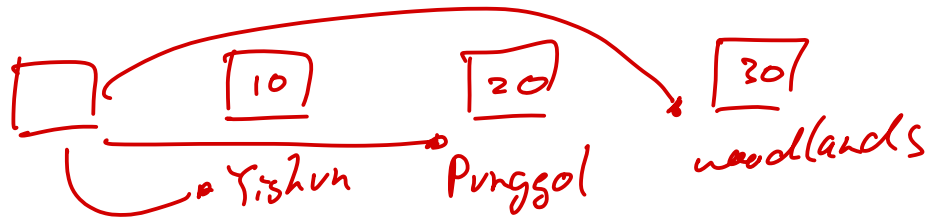
Reminders (PE 1)

- Date: 3 October 2020 (Saturday)
- Time: 9am to 12noon
- Duration: **2 hours 30 minutes**
- Online via ssh into restricted PE hosts
- 10% of your grade
- **Scope:** same as midterm
- **Format:** Five programming problems
- **Open Book** (printed/written notes only)
- Calculator allowed (but not needed)
- Criteria: *correctness, style, and efficiency*

Important details

- <https://mysoc.nus.edu.sg/academic/e-exam-sop-for-students/>
- <https://nus-cs1010.github.io/2021-s1/slides/pe1.md>

Pointers



- Pointers are a data type that stores an address of another variable
 - Analogy: A passerby on the street pointing to an address of a building
- &** - "address-of" operator
- *** - "dereference" operator ("location-of-address" operator)
- Pointers are like any other variable, they have their own address and memory space
- Special type of pointer
 - NULL pointer**, which points to nothing (Pointing to memory location of 0)

?? $c = \text{null}$

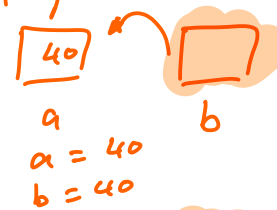
$\text{ptr} == \text{Null}$

$\text{long } a = 40;$
 $\text{long } b = a;$

$a = 40$
 $b = \text{copy of } a = 40$



$\text{long } a = 40;$
 $\text{long } *b = \&a;$



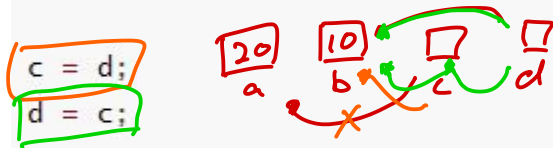
$a = 50$
 $b = 50$

$\text{fourty} = 40$
 $\&\text{fourty} = \text{a} \times 000B00A$

$\text{long } * \text{ptr};$
 $\text{ptr} = \&\text{fourty};$
 $\text{ptr} = \text{a} \times 000B00A$
 $\&\text{ptr} = 40$

Thinking time!

```
double a = 20;  
double b = 10;  
double *c = &a;    *c = 20  
double *d = &b;    *d = 10
```



```
cs1010_println_double(*c);    20  
cs1010_println_double(*d);    10
```

//what will be the output here?

```
printf("%p", <pointer here>);
```

Thinking time!

①


```
double a = 10;  
double *b = &a; //Is this legal? ✓ legal
```

error →  $\leftarrow \text{previous} = \frac{0 \times 11243}{20000}$

②

```
double a;  
*a = 20000; //Is this legal?
```

✓ legal, but will have segmentation error.



 ptr. 

③

```
double *a;  
a = 20000; //Is this legal?
```

✓ legal, but will throw error

④


```
double a; =   
double *b; =   
b = &a; //Is this legal?
```

*b = 40 // legal, no error.

⑤

```
double a;  
double *b;  
*b = &a; //Is this legal?
```

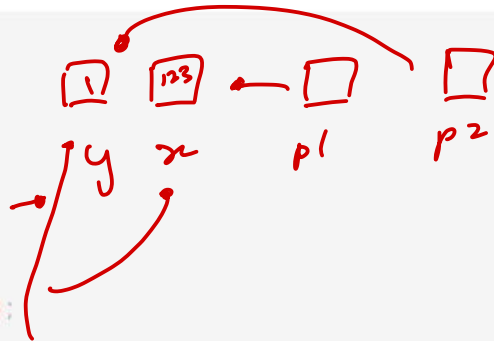
✓ legal, correct



Problem 14.1 Question

Sketch the content of the memory while tracing through the following code. What would be printed?

```
1 long *ptr1;
2 long *ptr2;
3 long x;
4 long y;
5
6 ptr1 = &x;
7 ptr2 = &y;
8
9 *ptr1 = 123;
10 *ptr2 = -1;
11
12 cs1010_println_long(x);
13 cs1010_println_long(y);
14 cs1010_println_long(*ptr1);
15 cs1010_println_long(*ptr2);
16
17 ptr1 = ptr2;
18 *ptr1 = 1946;
19
20 cs1010_println_long(x);
21 cs1010_println_long(y);
22 cs1010_println_long(*ptr1);
23 cs1010_println_long(*ptr2);
24
25 y = 10;
26
27 cs1010_println_long(x);
28 cs1010_println_long(y);
29 cs1010_println_long(*ptr1);
30 cs1010_println_long(*ptr2);
```



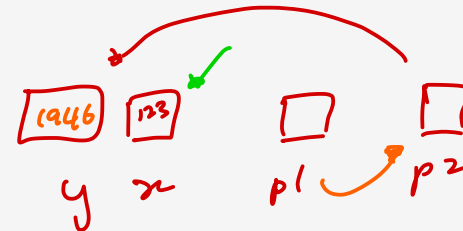
$\&x = \&y$

} (A)

123
-1
123
-1

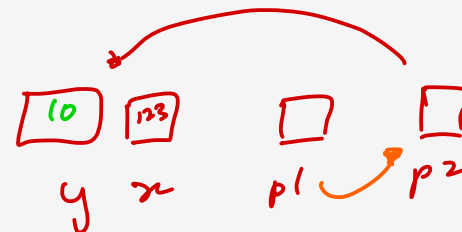
} (B)

123
1946
1946
1946



} (C)

123
10
10
10



Problem 14.1 Answer

This question reinforces the understanding pointers, the `*` and the `&` operator.

```
123
-1
123
-1
```

After changing `ptr1` to point to `ptr2` and changing `*ptr1`

```
123
1946
1946
1946
```

After changing `y` to 10:

```
123
10
10
10
```


Problem 14.2 Question

What is wrong with both programs below?

access
a pointer
that might
be freed

```
1 double *addr_of(double x)
2 {
3     return &x;
4 }
5
6 int main()
7 {
8     double c = 0.0;
9     double *ptr;
10
11     ptr = addr_of(c);
12     *ptr = 10;
13 }
```

$\text{print}(c) = 2.$

$x = 10$ $y = 10.$

if memory is used
by someone else.

= marked as free!

Read
a pointer
that might
be freed.

```
1 double *triple_of(double x)
2 {
3     double triple = 3 * x;
4     return &triple;
5 }
6
7 int main()
8 {
9     double *ptr;
10
11     ptr = triple_of(10);
12     cs1010_println_double(*ptr);
13 }
```

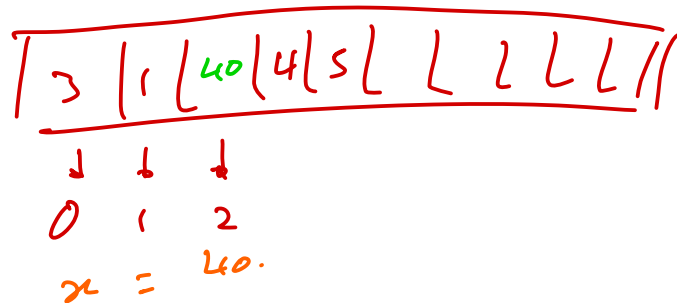
someB. window, mac.
Runtime.
 $\text{someB} + 2 \times 10 + 2 \times 10$
won't

Problem 14.2 Answer

- Note that both are calling functions that are returning pointers
 - Functions are created as instances, i.e. only created when needed, if not it will be “destroyed”
- After the function call, if we want to read and write to that memory address, it causes a dangerous issue as the memory for the function is returned to the OS
- The first program writes to the memory location (which is more dangerous as we may modify something we do not mean to)
- The second reads from it (which is less dangerous but may lead to unexpected values being read).

Arrays

Array. →



- A data structure to store a collection of variable
- Analogy: A cabinet to store your stuff
- Declaration of array

```
long a[10]; //Declares an array of length 10 with the name of 'a' to store long values
```

- Writing to array

```
a[2] = 40; //Setting the 3rd slot in the array 'a' to value of 40
```

- Accessing an array

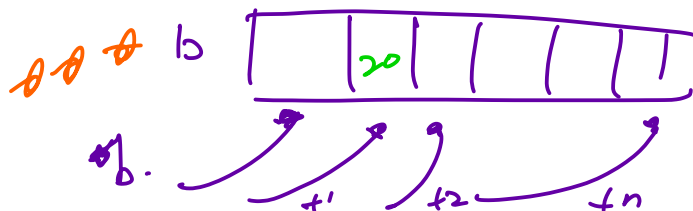
```
long x = a[2]; //Sets x to the value of the 3rd slot in the array 'a'
```

- Passing an array to a function

```
void foo(long a[]) { . . . }  
//or  
void foo(long a[10]) { .. }  
// foo accepts an array as argument (10 is for human understandability only)
```

**Note that index of array starts from 0*

Arrays



- Creating a new array with pointers

```
long *b[10];
```

//Declares an array of length 10 with the name of 'b' (Note that since this is declaration by pointer, calling 'b' points to the b[0], i.e. the first index of the array)

- Writing to array using pointers

```
*(b + 1) = 20; //Writes 20 to the 2nd position of the array 'b'
```

- Accessing array using pointers

```
long x = *b; //Writes the value of b[0] to long x
```

```
long y = *(b + 1) //Writes the value of b[1] to long y
```

- Using CS1010 library to create array

```
long *a = cs1010_read_long_array(10);
```

- Note that this is wrong

```
long *a = {1, 2, 3};
```

✗.

Problem 15.1 Question & Answer

Write the function `average` that takes an array of k integers and k and returns the average of the k values in the array.

```
double avg(const long list[], long length)
{
    long sum = list[0];
    for (long i = 1; i < length; i += 1) {
        sum += list[i];
    }
    return sum*1.0/length;
}

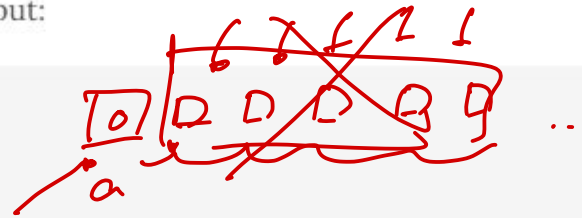
int main()
{
    long a[10] = {1, 2, 3, 4, 1, 9, 10, 44, -1, -5};
    cs1010_println_long(avg(a, 10));
}
```

Problem 15.2 Question

Explain why the following would lead to senseless output:

```
1 int main()
2 {
3     long a = 0;
4     cs1010_println_long(max(&a, 10));
5 }
```

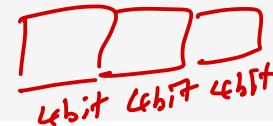
Array decay!



How about the following? Would the output be correct?

```
1 int main()
2 {
3     long a = 0;
4     cs1010_println_long(max(&a, 1)); // change 10 to 1
5 }
```

int → 4 bit-
double → 8 bit-
char → 8 bit-



~~ptr.~~
long max(long list[], long length)

```
{
    long max_so_far = list[0];
    for (long i = 1; i != length; i += 1) {
        if (list[i] > max_so_far) {
            max_so_far = list[i];
        }
    }
    return max_so_far;
}
```

Problem 15.2 Answer

This question is about passing in array to a function and about array decay.

In the function `max`, we only receive a pointer, when we access `list[2]`, for instance, we are accessing `*(&a + 2)`, so we will be accessing some unintended strange memory location.

Changing the length of the array to 1 would fix the problem since we will only access `list[0]` within `max`.

Problem 15.3 Question

Explain how the following code iterates through every element in the list when called with an array of length `length` as the first argument.

```
1 long max(long *list, long length)
2 {
3     long max_so_far;
4     long *curr;
5
6     max_so_far = *list;
7     curr = list + 1;
8     for (long i = 1; i != length; i += 1) {
9         if (*curr > max_so_far) {
10             max_so_far = *curr;
11         }
12         curr += 1;
13     }
14     return max_so_far;
15 }
```

Handwritten annotations:

- Line 3: *// Assign first value of list to max*
- Line 6: *max = list[0]*
- Line 7: *curr = list[1]*
- Line 8: *i != length* (circled)
- Line 9: *if (*curr > max_so_far)* with boxes around **curr* and *max_so_far*, and *curr* and *max* below.
- Line 12: *curr += 1* (circled) with an arrow pointing to *curr curr+1* below.

Handwritten notes:

- end.*
- max = list* (with a bracket pointing to the list elements below)
- Table of values:

<i>curr</i>	<i>i</i>	<i>length</i>
<i>list + 1</i>	<i>1</i>	<i>1</i>
<i>list + 2</i>	<i>2</i>	<i>2</i>
<i>list + 3</i>	<i>3</i>	<i>3</i>
<i>list + 4</i>	<i>4</i>	<i>4</i>

Additional notes: *list + 4* is crossed out with a red X. There is a double equals sign *==* between the *i* and *length* columns for the last row.

Problem 15.3 Answer



The key to understanding this is through variable `curr`:

- `*curr` is similar to `list[i]`
- `curr += 1` is similar to `i += 1`

Process of program writing

- Important steps for writing a program
 - Identifying what variables are needed
 - Breaking down the problem into smaller ones
 - Coming up with the idea to solve the problem
 - Coming up with flowcharts (or other forms to present the solution)
 - Check that the solution is correct, and
 - Only finally, write out the code in C *⊗ last step.*

Accessing PE accounts

- Email from prof
- Test log in
- Should see exercise 3 as skeleton
- Note that the account will be wiped after Thursday 6pm and will be inaccessible after that until PE

Assignment 3

- Efficient use of array
 - Direct access of array at index of input
 - Avoid using nested loops unless absolutely necessary
 - *Often can directly access array using user input*
- Be careful when passing array into another function
 - Array decay might happen
- Avoid using **variable sized array**
- Take note of value initialising when using array
 - Special cause of initialising using `a[100] = {0};`
 - `a[100] = {1};` // This only initialises the first index to 1, not the entire array
- Remember to `free()` after you are done with the array

$a[100] = \{1, 2, 3, 4, 5, \dots\}$
 $a[100] = \{0, \dots\} \rightarrow a[100] = \{0\};$

PE1 18/19 Questions

1 Vote (4 marks)

The Planet Earth is having its first democratic election to elect the first President of the World, from two candidates, McNeal and Nixon.

Write a program `vote` that reads, from the standard input, two integers representing the number of votes received by McNeal and Nixon in that order, and print, to the standard output, the percentage of votes received by the two candidates, in the same order.

You can assume that there is at least one vote in the election.

PE1 18/19 Questions

2 Newton (4 marks)

Newton's method is a numerical method used to determine the root of a mathematical function $f(x)$. In other words, it finds an x such that $f(x)$ is 0. It starts with an initial guess x_0 of a root of $f(x)$, and successively calculates x_i by:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)},$$

where $f'(x_i)$ is the derivative of $f(x)$. This process continues until $f(x_n)$ is close enough to the root.

In this question, you will write a program that calculates the root of a cubic function $f(x) = ax^3 + bx^2 + cx + d$ using the Newton's method. Recall that, $f'(x) = 3ax^2 + 2bx + c$.

Write a program `newton` that reads, from the standard input, five real values corresponding to a , b , c , d , and x_0 , in that order. Print to the standard output the root of the cubic function as obtained by Newton's method starting from the initial guess x_0 . For the purpose of this question, we say that x is close enough to a root if $|f(x)| < 0.000000001$.

Example

Suppose we have the input `1 2 0 5 -3`. We wish to find the root to the polynomial $f(x) = x^3 + 2x^2 + 5$ starting with $x_0 = -3$. The derivative $f'(x)$ is $3x^2 + 4x$. Executing the Newton's method, we get the following:

i	$f(x_i)$	$f'(x_i)$	x_{i+1}
0	-4.0000	15.0000	-2.7333
1	-0.4788	11.4800	-2.6916
2	-0.0107	10.9680	-2.6906
3	-0.0000	10.9562	-2.6906

Since $f(x_4)$ is close enough to 0, we output x_4 (-2.6906) as the root.

PE1 18/19 Questions

3 Goldbach (6 marks)

The Goldbach Conjecture is one of the oldest unsolved problems in mathematics. It states that "every even number is the sum of two prime numbers". This conjecture is not yet proven to be true for all even numbers, but have been shown to be true for even numbers up to 400,000,000,000,000.

Write a program, `goldbach`, that, reads, from the standard input, an even number n , and prints, to the standard output, the number of pairs of prime numbers where their sum is n . For example, 10 is the sum of the prime pair (3, 7) and (5, 5). So your program `goldbach` should print 2 when the input is 10. 100 is the sum of the prime pair (3, 97), (11, 89), (17, 83), (29, 71), (41, 59), and (47, 53) So your program should print 6 when the input is 100.

Note: A prime number is an integer larger than 1 that can only be divisible by 1 and itself.

PE1 18/19 Questions

4 Digits (8 marks)

An integer consists of multiple consecutive sequences of digits. For instance, the number 800100022, has five sequences, 8, 00, 1, 000, 22. The sequence 000 is the longest.

Write a program `digits` that reads, from the standard input, an integer, and prints to the standard output, the digit that occurs in the longest consecutive sequence. If there are multiple consecutive longest sequences, break ties by printing the digit that is the smallest.

You must not use arrays to solve this problem, or you risk getting 0 marks for this question.

PE1 18/19 Questions

5 Square (8 marks)

Write a program `square` that reads, from the standard input, an integer n ($n < 200$) and prints, to the standard output, a set of squares with width $n, n - 4, n - 8, ..$ until we reach either the width of 3, 2, 1, or 0. The smaller square is contained in the larger squares. The squares do not touch each other. A square has exactly one space between itself and the next larger square (if exists), in each direction.

You must not use arrays to solve this problem, or you risk getting 0 marks for this question. The only arrays you are allowed to use are in the form of strings `"#"` and `" "`.

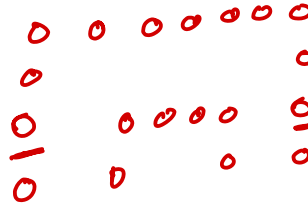
You must draw the squares *recursively*, by defining a recursive function that matches the following declaration:

```
void draw_square(long row, long width);
```

to draw the `row`-th row of the square with width `width`. As an example of how this function is called, the `main()` function has been given and it looks like:

```
#include "cs1010.h"
```

```
int main()
{
    long n = cs1010_read_long();
    for (long i = 0; i < n; i += 1) {
        draw_square(i, n);
        cs1010_println_string("");
    }
}
```



1 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 1

.

PE1 18/19 Answer

- <https://github.com/Khenus/CS1010Exercises>