CS1010 Tutorial 8

Slides by ***Ryan Tan Yu***

Agenda for Today

- Problem Set 20
- Problem Set 21
- Problem Set 22
- Assignment 5 Comments
- Assignment 6 Help

Problem Set 20

Ternary Expression `?`

- Denoted by the `?` operator
- `BOOLEAN_EXPRESSION ? VALUE_IF_TRUE : VALUE_IF_FALSE`
- Examples
  - `long i = x < 10 ? x : -1` -- assign `x` if `x < 10`, otherwise assign `-1`
  - `return ptr == NULL ? '\0' : ptr[0]`
    - return nul-terminating char if `ptr == NULL`, otherwise return `ptr[0]`
- A common way to write ternary expressions to closely follow `if...else` statements:

```
min = x < y       if (x < y) {
    ? x               min = x;
    : y;          } else {
                      min = y;
                  }
```

Problem Set 20.1(a)

Consider the macro below

```c
#define MIN(a,b)  a < b ? a : b

long i = MIN(10, 20);
long j = MIN(10, 20) + 1;
```

What are the values of `i` and `j` after executing the above?

Problem Set 20.1(a)

```
#define MIN(a,b)  a < b ? a : b

long i = MIN(10, 20);
long j = MIN(10, 20) + 1; // expected to be 11
```

expands to

```
long i = 10 < 20 ? 10 : 20;      // 10
long j = 10 < 20 ? 10 : 20 + 1;  // 10
```

- `j` is wrongly `10` despite the pre-expanded code looking "correct"
- **Preprocessor macros are just textual replacements**

Pre- and Post-increment Operator

- For integer types, increment a variable using `i++` or `++i`
- Post-increment `i++`
  - Current value of `i` is used for the current statement
  - `i` is incremented after
- Pre-increment `++i`
  - `i` is incremented then the value is used in the current statement

```
long x = 10;
long y = x++;
// {x == 11  &&  y == 10}

long x = 10;
long y = ++x;
// {x == 11  &&  y == 11}
```

Problem Set 20.1(b)

Consider the following code:

```
#define MIN(a,b)  a < b ? a : b

long i = 10;
long j = 20;
long k = MIN(j, i++);
```

What are the values of `i` and `k` after executing the above?

Problem Set 20.1(b)

```
#define MIN(a,b)  a < b ? a : b

long i = 10;
long j = 20;
long k = MIN(j, i++);
```

- What is the expected behaviour?

```
long i = 10;
long j = 20;
long k = MIN(j, i++); // {k == min(i,j) == i == 10}
// {i == 11}
```

Problem Set 20.1(b)

```
#define MIN(a,b)  a < b ? a : b

long i = 10;
long j = 20;
long k = MIN(j, i++);
```

- What is the actual behaviour?

```
long i = 10;
long j = 20;
long k = j < i++ ? j : i++; // {k == i + 1 == 11}
// {i == 12}
```

Problem Set 20.1(b)

```
long i = 10;
long j = 20;
long k = j < i++ ? j : i++; // {k == i + 1 == 11}
// {i == 12}
```

- `i++` returns the value of `i` for use in the expression, then increments `i`
- The program checks for `j < i++`
  - `10` is returned for use in `j < i++`
  - `i` is incremented to `11`
- `j < i++` checks for `20 < 10`
- The "false" branch of the ternary expression is taken
  - `i++` is evaluated again, `11` is returned and assigned to `k`
  - `i` is incremented to `12`
- Therefore, `{i == 12}` and `{k == 11}`

Problem Set 20.2

```
#define SWAP(T, x, y) T temp = x;\
  x = y;\
  y = temp;
```

What could go wrong if we write an `if...else` block without braces, along with the above macro?

Problem Set 20.2

```
#define SWAP(T, x, y) T temp = x;\
  x = y;\
  y = temp;

long large = 1;
long small = 100;
if (large < small)
    SWAP(long, large, small)
```

expands to

```
long large = 1;
long small = 100;
if (large < small)
    long temp = large;
larger = small;
small = temp; // error, temp not declared
```

Problem Set 20.2

- Avoid using `if...else`, `while`, etc blocks without braces
- Avoid using block macro-functions unless you know what you're doing
- A common idiom for multi-line macros is to wrap it in a `do { ... } while(0)`
- ***Optional*** Read up on why `do...while(0)` is a common macro idiom and *not* just wrapping the statements in braces `{...}`

```
#define FREE_2D(ptr, n)                                 \
    do {                                                \
        for (long __i = 0; __i < n; __i += 1) {         \
            free(ptr[__i]);                             \
        }                                               \
        free(ptr);                                      \
    } while (0)
```

Problem Set 21

Problem 21.1

```
void foo(long x) {
    if (x % 2 == 0) {
        // do something
    } else {
        assert(x % 2 == 1);
    }
}
```

- Will the assertion ever fail?
    - i.e will the expression in the assert statement ever be false?

Problem 21.1

```c
void foo(long x) {
    if (x % 2 == 0) {
        // do something
    } else {
        assert(x % 2 == 1);
    }
}
```

- Looks pretty clear cut
  - A number is either even or odd (parity), and the expression `n % 2 == ???` checks for parity
  - If `x % 2 == 0` is false, then `x % 2 == 1` must be true
  - ...Right?

Problem 21.1

```
void foo(long x) {
    if (x % 2 == 0) {
        // do something
    } else {
        assert(x % 2 == 1);
    }
}
```

- What if `x` is negative?
- `x % 2 == 1` **always returns 0** for all negative `x`
- Therefore, the assertion will fail
- **Even though the assertion may look obviously true, it's not always the case**

Problem Set 22

Problem 22.1

Order the following functions in **increasing** order of rate of growth

The functions are:

$$n! \qquad 2^n \qquad \log_{10} n$$

$$\ln n \qquad n^4 \qquad n \ln n$$

$$n \qquad n^2 \qquad e^n \qquad \sqrt{n}$$

Problem 22.1

- We can convert any instance of specialised logarithms (i.e to a certain base) using a multiplicative constant
- $\log_{10} n$ and $\ln n$ are the same - denote either as $\log n$
- Rank the rest via the three classes - sub-polynomial, polynomial, exponential
  - $\log n \quad \sqrt{n}$
  - $n^4 \quad n \log n \quad n \quad n^2$
  - $n! \quad 2^n \quad e^n$

Problem 22.1 - Sub-polynomial

$$\log n \qquad \sqrt{n}$$

- An easy way to guess is to try some large value of n
- For logarithms, using either base 2, e or 10 is reasonable
  - $\log_{10}(1 \times 10^6) = 6$
  - $\sqrt{1 \times 10^6} = 1000$
- Both functions are always increasing (*monotonically increasing*)
- Conclusion: $\sqrt{n}$ grows faster than $\log n$, therefore $\log n < \sqrt{n}$

Problem 22.1 - Polynomial

$$n^4 \qquad n \log n \qquad n \qquad n^2$$

- Obviously, $n < n^2 < n^4$
- Where does $n \log n$ fit in?
  - $\log n < n$
  - Therefore $n < n \log n < n^2$
- Conclusion: $n < n \log n < n^2 < n^4$

Problem 22.1 - Exponential

$$2^n \qquad e^n \qquad n!$$

- $2 < e \implies 2^n < e^n$
- Intuitively, $n! > e^n$ for some $n > k_0$ where $k \in \mathbb{N}$
- Therefore, $2^n < e^n < n!$

Problem 22.1

Solution:

$$\log n < \sqrt{n} < n < n \log n < n^2 < n^4 < 2^n < e^n < n!$$

Problem 22.2

- Given a code snippet, state its Big-O running time in terms of n
- Note
  - Printing a number is $O(1)$
  - Any arithmetic is $O(1)$

Problem 22.2(a)

State the Big-O running time of the following code, in terms of n

```
for (long i = 0; i < n; i += 1) {
    for (long j = 0; j < n; j += 2) {
        cs1010_println_long(i + j);
    }
}
```

- Printing a number is $O(1)$
- Any arithmetic is $O(1)$

Problem 22.2(a)

State the Big-O running time of the following code, in terms of n

```
for (long i = 0; i < n; i += 1) {
    for (long j = 0; j < n; j += 2) {
        cs1010_println_long(i + j);
    }
}
```

- The outer loop runs n times
- The inner loop runs $\frac{n}{2}$ times

$$f(n) = \frac{n}{2} \cdot n = \frac{n^2}{2} \in O(n^2)$$

Problem 22.2(b)

State the Big-O running time of the following code, in terms of n

```
for (long i = 1; i < n; i *= 2) {
    for (long j = 1; j < n; j *= 2) {
        cs1010_println_long(i + j);
    }
}
```

- Printing a number is an $O(1)$ operation
- Arithmetic is $O(1)$

Problem 22.2(b)

```
for (long i = 1; i < n; i *= 2) {
    for (long j = 1; j < n; j *= 2) {
        cs1010_println_long(i + j);
    }
}
```

- The outer loop runs $\log n$ times
- The inner loop runs in $\log n$ times

$$f(n) = \log n \cdot \log n = \log^2 n \in O(\log^2 n)$$

Problem 22.2(c)

State the Big-O running time of the following code, in terms of n

```
long k = 1;
for (long j = 0; j < n; j += 1) {
    k *= 2;
    for (long i = 0; i < k; i += 1) {
        cs1010_println_long(i + j);
    }
}
```

- Printing a number is an $O(1)$ operation
- Arithmetic is $O(1)$

Problem 22.2(c)

```
long k = 1;
for (long j = 0; j < n; j += 1) {
    k *= 2;
    for (long i = 0; i < k; i += 1) {
        cs1010_println_long(i + j);
    }
}
```

- The outer loop runs n times
- How many times does the inner loop run?
  - It seems to depend on the number of times the outer loop has run
  - **At any point in the program, the number of iterations of the inner loop is given by 2^{j+1}**

Problem 22.2(c)

```
long k = 1;
for (long j = 0; j < n; j += 1) {
    k *= 2;
    for (long i = 0; i < k; i += 1) {
        cs1010_println_long(i + j);
    }
}
```

- The total number of iterations is then given by

$$f(n) = \sum_{i=1}^{n} 2^i \approx 2^n \in O(2^n)$$

Food for Thought

Primality Testing

- To date, there is only one known *polynomial time* algorithm for primality testing
  - AKS Primality Test that runs in $O(\log^6 n)$ time
- The `is_prime` algorithm discussed in class seems to run in $O(\sqrt{n})$ time
  - Each iteration runs in $O(1)$ assuming division is $O(1)$
  - Even if division is not $O(1)$, it's still polynomial in the length of the input given the long division algorithm, therefore the overall algorithm will still be polynomial
- Why is it not considered a polynomial time algorithm?

Food for Tought (cont.)

- What does Big-O represent?
- What is the "length" of an input of a number?

Assignment 5 Comments

Contact, Social

- Do try to practice better code style
  - Avoid using short, non-descriptive variable names e.g (`m`, `flag` or `count1` and `count2`)
  - Avoid very long functions
  - Avoid deep nesting of `for` loops (more than 2 nested loops calls for a new function)
- Try to break down the problem into more sub-problems

Common Mistakes (Contact, Social)

- The algorithm was generally understood
- More abstraction could have been done
  - `is_friend(char** network, long i, long j)`
  - `set_friend(char** network, long i, long j)`
  - `copy_network(char **dest, char **src, long n)`
  - `init_network(long n)`
  - …

Common Mistakes (Contact, Social)

- Modifying the network *in-place* introduces bugs
- You should always leave the current degree matrix in-tact, and assign to a newly allocated matrix
    - Repeat for each computed degree

Common Mistakes (Life)

- Updating the `world` in-place leads to errors
- For example
    - `world[2][2]` depends on the state of `world[1][1]`, but if `world[1][1]` changes, `world[2][2]` will be incorrect
- Should make a new `world` and then populate it using the old world
- Then just swap the old and new world

Miscellaneous

- The CS1010 I/O library has a function `char** cs1010_read_word_array(long)`
  - Can be used to read the 2-D jagged matrix

Memory Issues

- `NULL` returns from `calloc`, `malloc`, `read_*_array(long)`, etc should be checked *immediately after* each call

```
long n = cs1010_read_long();
char *s = cs1010_read_word();
if (s == NULL) { // check IMMEDIATELY after
    cs1010_println_string("NULL return from read_word(), exiting...");
    exit(1);
}
```

Memory Issues

- Assigning to a pointer that points to memory allocated `malloc` or `calloc` without a `free` causes a memory leak

```
long n = cs1010_read_long();
char *s = malloc(n + 1);

// no call to free()

s = malloc(1); // memory leak!
```

- Any CS1010 I/O library function that returns a pointer uses `malloc` and `calloc` internally and the memory must be freed!

Memory Issues

- Memory must be *completely* freed

```
long n = cs1010_read_long();
char **arr = cs1010_read_word_array(n);

// ...

for (long i = 0; i < n; i++) {
    free(arr[i]); // free each row
}
free(arr); // free the array of pointers
```

Memory Issues (cont.)

- Here's a helpful macro to free a 2-D array

```
#define FREE_2D(arr, len)                              \
  do {                                                 \
    for (long __i = 0; __i < len; __i += 1) {   \
      free(arr[__i]);                                  \
    }                                                  \
    free(arr);                                         \
  } while (0)
```

Assignment 6 Hints

Some Encouragement

- **This assignment is not as difficult as Assignment 5**
- Don't feel discouraged by AS5
  - It's actually quite tough
- Try to do all the questions in this one

Add

- You have to code out the addition algorithm you learnt in Primary School
- The hard part is managing the size of the array and indexing
- If you want to make indexing easier
    - Implement a function `void reverse(char* s)` that reverses a given string
- Always take note of the carry-over from the previous addition

	1		1	1	
	9	3	4	3	2
		9	1	6	9
1	0	2	6	0	1

Frequency

- $O(n)$ time means you can only make a constant number of passes of each string
- For this question, you only need to make a *single* pass through both strings
- **Hint: it's very similar to Counting Sort**

Permutation

- If you can't figure out the $O(n+k)$ algorithm, just implement the fastest algorithm you can think of
- The $O(nk^2)$ algorithm
  - Let h be the "haystack", and s be the "needle"
  - For each substring, h_i of length $|s|$ in h
    - Use a double `for`-loop to check if h_i is a permutation of s
  - Outer loop runs n times, the inner loop runs in $O(k^2) \implies O(nk^2)$
- The hint given is to use the code written in the previous question for optimisation
- Hint for $O(n+k)$ - *sliding window*