CS1010 Laboratory 08
Zhang Puyu
Exercise 5 Review
Read in Character Matrices
Frequently Seen Bugs
How to Solve Last 3???
C Pre-processing
Bonus Info
Searching And Sorting
Exercise 6

CS1010 Laboratory 08

Searching, C-Preprocessing, Sorting, Exercise 6

Zhang Puyu

Group BD04

October 25, 2024

Plan of the Day

Exercise 5 Review

Read in Character Matrices

Frequently Seen Bugs

How to Solve Last 3???

C Pre-processing

Bonus Info

Searching And Sorting

Exercise 6

1 Exercise 5 Review
- Read in Character Matrices
- Frequently Seen Bugs
- How to Solve Last 3???

2 C Pre-processing

3 Bonus Info

4 Searching And Sorting

5 Exercise 6

Easier Ways to Read in Character Matrices


```c
size_t n = cs1010_read_size_t();
char **mat = calloc(n, sizeof(char *));
if (mat == NULL) {
    // Deal with error
}
for (size_t i = 0; i < n; i += 1) {
    mat[i] = cs1010_read_word();
    if (mat[i] == NULL) {
        // Deal with error
    }
}
```

is the same as

```c
size_t n = cs1010_read_size_t();
char **mat = cs1010_read_word_array(n);
if (mat == NULL) {
    // Same error handling procedures
}
```

Easier Ways to Read in Character Matrices


```c
size_t n = cs1010_read_size_t();
char **mat = calloc(n, sizeof(char *));
if (mat == NULL) {
    // Deal with error
}
for (size_t i = 0; i < n; i += 1) {
    mat[i] = cs1010_read_line();
    if (mat[i] == NULL) {
        // Deal with error
    }
}
```

is the same as

```c
size_t n = cs1010_read_size_t();
char **mat = cs1010_read_line_array(n);
if (mat == NULL) {
    // Same error handling procedures
}
```

Difference between `read_word` And `read_line`

`read_word` will parse everything **before the first white space or line break**. This implies that `read_word_array` splits elements by both white spaces and line breaks.

Whereas `read_line` will take everything **up to the first line break**, i.e., the line break character `'\n'` will be read in also. This implies that `read_line_array` splits elements by line breaks only.

Question: `word_array` or `line_array`?

- `abc def opq ijk`
- **Answer:** Use `word_array`
- ```
  abcd
  ijk
  mn
  opq
  xyzw
  ```
- **Answer:** Either one works but take note of the line break characters.
  ```

Overwriting A Dynamic Array


```c
int main() {
    size_t n = cs1010_read_size_t();
    char **string = calloc(n, sizeof(char *));
    for (size_t i = 0; i < n; i += 1) {
        string[i] = calloc(i + 1, sizeof(char)); // Allocate row
    }
    for (size_t i = 0; i < n; i += 1) {
        string[i] = cs1010_read_word(); // Read in input
    }
    for (size_t i = 0; i < n; i += 1) {
        free(string[i]);
    }
    free(string);
}
```

Question: Why Memory Leak?

- Each row `string[i]` is **allocated twice**: once by `calloc` and once by `cs1010_read_word`.
- The second allocation **overwrites** the pointer from the previous allocation.
- So the memory space allocated by `calloc` becomes **unreachable!**

Copy A Dynamic Array

CS1010 Laboratory 08

Zhang Puyu

Exercise 5 Review

Read in Character Matrices

Frequently Seen Bugs

How to Solve Last 3???

C Pre-processing

Bonus Info

Searching And Sorting

Exercise 6

```c
int main() {
    size_t n = cs1010_read_size_t();
    char *a = cs1010_read_word(); // Read a string of length n
    char *b = calloc(n, sizeof(char));
    for (size_t i = 0; i < n; i += 1) {
        b[i] = a[i]; // Copy a into b
    }
    free(b);
}
```

Question: Why Memory Leak?

- Note that a and b are **distinct arrays** despite having identical values!
- So we still have to free a after freeing b.

Consume A Dynamic Array

CS1010 Laboratory 08

Zhang Puyu

Exercise 5 Review

Read in Character Matrices

Frequently Seen Bugs

How to Solve Last 3???

C Pre-processing

Bonus Info

Searching And Sorting

Exercise 6

```c
char *consume(char *string, size_t n) {
    return calloc(n, sizeof(char));
}
int main() {
    size_t n = cs1010_read_size_t();
    char *a = cs1010_read_word(); // Read a string of length n
    a = consume(a, n);
    free(a);
}
```

Question: Why Memory Leak?

- Note that `a` becomes the returned pointer from `consume` after the function call! I.e. `a` gets **consumed** by the function and replaced by another pointer.
- So `free(a)` does not free the **original pointer** `a`.

`line.c`: Not A Bug, But A Feature


```
for(long x = x1; x <= x2; x += 1) {
    double y = (double)y1 + m * (double)(x - x1);
    double rounded_y = round(y);
    canvas[(long)rounded_y][x] = 255;
}
```

Anything wrong here? **No!** But this code will not pass test case 4, why? The test cases are designed for floating point arithmetic, which cause a slight precision mismatch if we use `long` here.

In test case 4, at $x = 10$, y evaluates to 6.5000 (you may print to verify). So `round(y)` will be 7. But in reality the correct value is 6.

`line.c`: Not A Bug, But A Feature

The following, however, works fine:

```
double y = (double)y1;
for(long x = x1; x <= x2; x += 1) {
    double rounded_y = round(y);
    matrix[(long)rounded_y][x] = 255;
    y += m;
}
```

At $x = 10$, y is still *displayed as* 6.5000, but due to the imprecision of `double` it is actually 6.4 something (extremely close to 6.5). Thus, after rounding it is 6 as intended.

This is **not a real bug**, but **a feature** of `double`.

contact.c: Be Careful with Triangular Matrices


```
bool are_direct_contacts(long i, long j, char **contacts) {
    // contacts is a lower triangular matrix
    return contacts[i][j] == '1' || contacts[j][i] == '1';
}
```

Discuss: Will this work?

NO! In a lower triangular matrix, all entries (x, y) with $y > x$ do not exist! So if we happen to call this function with $j > i$, we will get **heap buffer overflow!**

popular.c: How to Count

The "naïve" way:

```
char *count_friends(char **map, long k, long n) {
    long count = 0;
    for (long i = 0; i < k; i += 1) {
        if (map[k][i] == '1') {
            count += 1;
        }
    }
    for (long j = k + 1; j < n; j += 1) {
        if (map[j][k] == '1') {
            count += 1; // Don't forget those with bigger IDs
        }
    }
    return count;
}
```

Discuss: Are the `if`-conditional checks really necessary?

Hint: We know how to convert a digit character into its corresponding integer.

popular.c: How to Count

A (possibly) more elegant way:

```
char *count_friends(char **map, long k, long n) {
    long count = 0;
    for (long i = 0; i < k; i += 1) {
        count += (map[k][i] - '0');
    }
    for (long j = k + 1; j < n; j += 1) {
        count += (map[j][k] - '0');
    }
    return count;
}
```

Note: This does not improve efficiency, but is just a slightly more elegant implementation in my opinion.

`social.c`: Difficult!...?

I personally call such type of questions "generative problems".

In such problems, we are usually given some parameter $p(n)$ and the input as the case for $p(1)$ (or $p(0)$), and tasked to derive the case for $p(k)$.

The natural thing to do is to **build up** the cases step by step, i.e.,

$$1 \to 2 \to 3 \to 4 \to \cdots \to k.$$

The core of the solution to these problems is: what is the **procedure** for us to systemically generate $p(n+1)$ from $p(n)$?

social.c: Recursive Approach

- Let $M(n)$ be the relation mapping of degree n. Our input is $M(1)$ and we want to find $M(k)$.
- Find $M(k-1)$ first.
- Now consider i, j to be any two distinct persons.
  - i, j are related in $M(k-1) \implies$ they are related in $M(k)$ (trivial).
  - Otherwise, i, j are not related in $M(k-1)$.
    - If there is some $k \neq i, j$ such that i, k are related in $M(k-1)$ and j, k are related in $M(1)$ (or the other way round) $\implies$ i, j are related in $M(k)$.
    - Otherwise $\implies$ i, j are not related in $M(k)$.

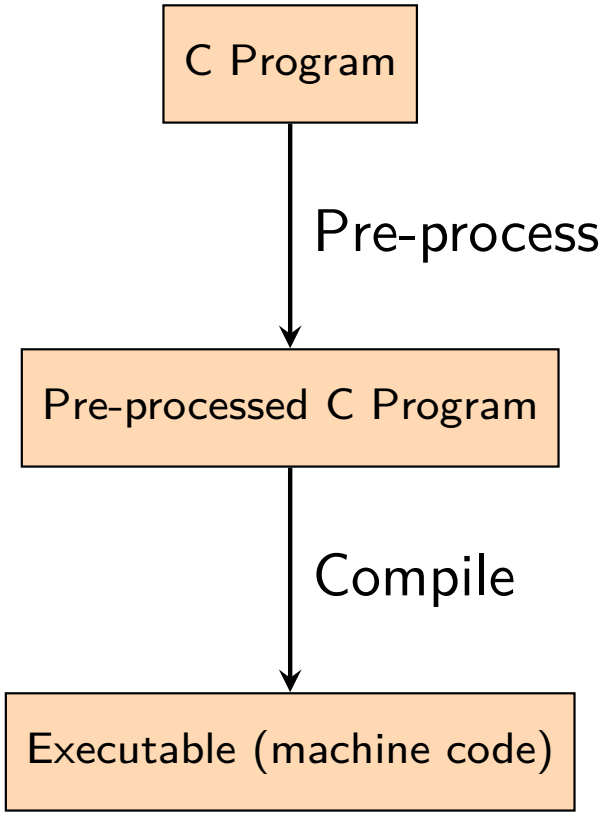
Remark. At any point in time, we have **3 different mappings**: $M(1)$ and $M(n-1)$ as reference mappings, and $M(n)$ as the current mapping we are constructing.

social.c: Recursive Approach

Sample code:

```
char **mapping(long deg, long num_ppl, char **base_map) {
    if (deg == 1) {
        // Base case. What to do here?
    }
    // Get previous mapping
    char **prev_map = mapping(deg - 1, num_ppl, base_map);
    // Allocate a dynamic matrix to store current mapping
    char **map = triangular_matrix(num_ppl);
    for (long i = 0; i < num_ppl; i += 1) {
        for (long j = i; j < num_ppl; j += 1) {
            if (prev_map[j][i] == FRIEND) {
                // TODO
            } else {
                // TODO
            }
        }
    }
    // Anything else before we return?
    return map;
}
```

C Compilation Pipeline



C Pre-processing: Analogies

CS1010 Laboratory 08

Zhang Puyu

Exercise 5 Review

Read in Character Matrices

Frequently Seen Bugs

How to Solve Last 3???

C Pre-processing

Bonus Info

Searching And Sorting

Exercise 6

Analogy 1: Defining an Equation

The population growth rate of an organism can be modelled with

$$\frac{\mathrm{d}N}{\mathrm{d}t} = rN\left(1 - \frac{N}{K}\right).$$

Issue: No one understands what each of the variables means!

How to rectify this? We can **pre-define** the meaning of each variable before stating the equation!

C Pre-processing: Analogies

Analogy 1: Defining an Equation

Let

- N be the current population size;
- t be the time;
- r be the natural rate of population growth;
- K be the maximal population which the environment can carry.

The population growth rate of an organism can be modelled with

$$\frac{\mathrm{d}N}{\mathrm{d}t} = rN\left(1 - \frac{N}{K}\right).$$

Now this equation makes more sense!

C Pre-processing: Analogies

Analogy 2: Table of Contents of Textbooks

Non-linear Programming

Table of Contents

- **Terminologies and Notations**
- Convexity
- Unconstrained Optimisation
- Constrained Optimisation
- Useful Numerical Methods

C Pre-processor

CS1010 Laboratory 08
Zhang Puyu
Exercise 5 Review
Read in Character Matrices
Frequently Seen Bugs
How to Solve Last 3???
C Pre-processing
Bonus Info
Searching And Sorting
Exercise 6

- **Pre-processor directives** are statements which serve similar purposes as the preamble to a book.
- These directives starts with # and define **external libraries, global constants** and **macros** which are used in the program.
- A **C pre-processor** reads these directives.
- Examples:
  - `#include "cs1010.h"`
  - `#include <math.h>`
  - `#define NROWS 10`
  - `#define ll long long`

Constants: Enhance Readability and Extensibility


```
for (long i = 0; i < 15; i += 1) {
    for (long j = 0; j < 27; j += 1) {
        sum += a[i][j];
    }
}
```

After a long time, the coder might forget what he/she meant when first writing the code: why 15 and 27?

(In some sense they are nice numbers as 15 is the product of the first two odd primes and 27 is the first odd prime raised to the power of itself...)

Such numbers are known as **magic numbers** and should be avoided as much as possible for the sake of readability.

Constants: Enhance Readability and Extensibility

CS1010 Laboratory 08

Zhang Puyu

Exercise 5 Review

Read in Character Matrices

Frequently Seen Bugs

How to Solve Last 3???

C Pre-processing

Bonus Info

Searching And Sorting

Exercise 6

```
#define NROWS 15
#define NCOLS 27
// Intermediate code omitted
for (long i = 0; i < NROWS; i += 1) {
    for (long j = 0; j < NCOLS; j += 1) {
        sum += a[i][j];
    }
}
```

Now the code looks more descriptive and the purpose of the loop is much clearer.

Suppose the above loop is used multiple times in the program. Using constants allows us to efficiently alter their values without editing every occurrence of them.

include: Build up the Dependency Tree

Program a.c:

```
#include "b.h"
int main() {
    foo(PI);
}
```

Program b.h:

```
#include "c.h"
#define PI 3.1416
```

Program c.h:

```
#include "cs1010.h"
void foo(double x) {
    cs1010_println_double(x);
}
```

What will happen if we compile with `clang -E a.c`?

Macros

A **macro** is a code snippet that is substituted into the program and expanded during pre-processing.

Example:

```
#include "cs1010.h"
#define SQUARE(x) x * x

int main() {
    // Will output 25
    cs1010_println_long(SQUARE(5));
    // Will output 16.0000
    cs1010_println_double(SQUARE(4.0));
    return 0;
}
```

Macros: Generic Types

CS1010 Laboratory 08
Zhang Puyu
Exercise 5 Review
Read in Character Matrices
Frequently Seen Bugs
How to Solve Last 3???
C Pre-processing
Bonus Info
Searching And Sorting
Exercise 6

We can use a **generic type** (or **type parameter**) to restrict the type of the arguments used in a macro.

Example:

```
#include "cs1010.h"
#define SWAP(T, x, y) {T t; t = x; x = y; y = t}

int main() {
    long a = 1;
    long b = 2;
    SWAP(long, a, b); // Now a == 2 && b == 1
    char m[4] = "abc";
    char n[4] = "123";
    SWAP(char *, m, n);
    // Now m is "123" and n is "abc"
    return 0;
}
```

Macros: Pitfall

Be careful with situations like this:

```
#include "cs1010.h"
#define SQUARE(x) x * x

int main() {
    cs1010_println_long(SQUARE(5 + 1));
    // The above gets expanded to 5 + 1 * 5 + 1
    return 0;
}
```

Therefore, we should always use brackets around the arguments of a macro, i.e., `SQUARE(x) (x) * (x)` is safe.

If-Defined Direvatives

- The `#ifdef` and `#endif` directives can be used to define alternative code for different compiler configurations.
- You can use this to segregate function bodies for different operating systems.

```
void which_os() {
#ifdef _WIN32
    cs1010_println_string("windows");
#endif
#ifdef __APPLE__
    cs1010_println_string("apple");
#endif
#ifdef __linux__
    cs1010_println_string("linux");
#endif
}
```

If-Defined Direvatives

CS1010 Laboratory 08

Zhang Puyu

Exercise 5 Review

Read in Character Matrices

Frequently Seen Bugs

How to Solve Last 3???

C Pre-processing

Bonus Info

Searching And Sorting

Exercise 6

You can also use this to define debug-mode-only versions of a function.

```
void some_function(long n) {
#ifdef DEBUG
    cs1010_print_string("Input is: ");
    cs1010_println_long(n);
#endif
    // remaining code here:
}
```

When compiled with the `-DDEBUG` flag, the code between the `#ifdef` and `#endif` directives will be included in the final executable.

Bonus Information

CS1010 Laboratory 08

Zhang Puyu

Exercise 5 Review

Read in Character Matrices

Frequently Seen Bugs

How to Solve Last 3???

C Pre-processing

Bonus Info

Searching And Sorting

Exercise 6

There are five major types of operations which are core to algorithm optimisation: **insertion**, **removal**, **retrieval**, **searching** and **sorting**.

In CS1010, we briefly introduce the basics of searching and sorting (more advanced stuff along with the other operations will be discussed in CS2040/S/C).

However, the various algorithms can be quite abstract and hard to visualise...

Now introducing **the BEST visualisation tool**: VisuAlgo, developed by our beloved(?) Prof Steven Halim in NUS Computing, with his textbook for competitive programming.

Binary Search

Probably the **most powerful** search algorithm for simple arrays.

An essential prerequisite: the array is **sorted**.

```
long binary_search(long *a, long left, long right, long v) {
    if (left > right) {
        return -1; // Not found
    }
    size_t mid = (left + right) / 2;
    if (a[mid] == v) {
        return mid;
    }
    if (a[mid] > v) {
        // Anything to the right of a[mid] must > v
        // So we only search the left half
        return binary_search(a, left, mid - 1, v);
    }
    // Anything to the left of a[mid] must < v
    // So we only search the right half
    return binary_search(a, mid + 1, right, v);
}
```

Binary Search

Now to convince you that binary search is way more efficient than linear search...

Proof.

Let $T(n)$ be the number of steps to perform binary search on an array of size n. Clearly

$$T(n) = T\left(\frac{n}{2}\right) + c, \qquad T(1) = t$$

where c and t are constants. The recurrence relation can be re-written as

$$T(n) - T\left(\frac{n}{2}\right) = c.$$

Now consider

$$\sum_{i=0}^{\log_2 n} \left[T\left(\frac{n}{2^i}\right) - T\left(\frac{n}{2^{i+1}}\right)\right] = T(n) - T(1) = c\left(\log_2 n + 1\right),$$

so

$$T(n) = c\left(\log_2 n + 1\right) + t \in \mathcal{O}\left(\log n\right).$$

□

Comparison-Based Sorting ($\mathcal{O}\left(n^2\right)$)

1. Bubble Sort
   - Iterate through the entire array.
   - For any `a[i] > a[i + 1]`, swap them.
   - After the k-th pass, we know that the **last k elements** of the array are sorted.
   - We need n passes to sort an array of size n, in the k-th pass we iterate $(n - k + 1)$ elements (arithmetic series).
   - **Discuss:**
     - Can we terminate the algorithm earlier?
     - Is there any case where bubble sort outperforms $\mathcal{O}\left(n^2\right)$?
2. Insertion Sort
3. Selection Sort

Comparison-Based Sorting ($\mathcal{O}\left(n^2\right)$)

1. Bubble Sort
2. Insertion Sort
   - Iterate through the entire array.
   - For every element, repeatedly swap it with its previous element, until we encounter an element less than or equal to it, or we have reached index 0.
   - In worst case, the k-th element will do k swaps (arithmetic series). **What is this worst case input?**
   - Do you realise this is actually how we solved `largest.c`?
   - **Discuss:** Is there any case where insertion sort outperforms $\mathcal{O}\left(n^2\right)$?
3. Selection Sort

Comparison-Based Sorting ($\mathcal{O}\left(n^2\right)$)

1. Bubble Sort
2. Insertion Sort
3. Selection Sort
   - At the k-th iteration, find the least element between `array[k - 1]` and `array[n - 1]`.
   - Swap this least element with `array[k - 1]`.
   - It is $\mathcal{O}(k)$ to find the minimum, and we need to repeat this for $k = 1, 2, \cdots, n$ (arithmetic series).
   - **Discuss:** Is there any case where selection sort outperforms $\mathcal{O}\left(n^2\right)$?
4. Other popular sorting algorithms: radix sort, merge sort, quick sort, heap sort (in CS2040/S/C).

Counting Sort

- We construct a **frequency table** for every integer that has appeared in the array.
- Iterate through the frequency table in ascending order.
- For integer i with frequency f_i, we will insert i into the next f_i empty slots in our output array.
- The output array will be sorted.
- Let the input array be of size n with maximum element k. We need $\mathcal{O}(n)$ to compute the frequency table and $\mathcal{O}(k)$ to iterate the frequency table. So overall it is $\mathcal{O}(n+k)$.
- If k is not very large, we see that counting sort is very fast.
- **Question:** So why not just use counting sort for everything?

Higher-Level Discussion of `sort.c`

CS1010 Laboratory 08

Zhang Puyu

Exercise 5 Review

Read in Character Matrices

Frequently Seen Bugs

How to Solve Last 3???

C Pre-processing

Bonus Info

Searching And Sorting

Exercise 6

- **Questions** to test your understanding of V-arrays:
  - Let $v[k]$ to be the **minimal** element of a V-array of size n, for $0 \leq i < k$, what can we say about $v[i]$ and $v[i+1]$?
  - How about $v[j]$ and $v[j+1]$ for $k \leq j < n-1$?
  - Are the followings valid V-arrays?
    - $1, 1, 1, 1, 1$
    - 9
    - $-1, 9, 10$
    - $10, 9, -1$
    - $1, 2, 5, 4, 3$
- Now **discuss:**
  - What is an (obvious) $\mathcal{O}\left(n^2\right)$ solution?
  - What is an $\mathcal{O}\left(n \log n\right)$ solution? (Hint: What happens when we split a V-array into 2 **sub-arrays**?)
  - How about an $\mathcal{O}(n)$ solution? (Play with some simple examples to get intuition.)

Higher-Level Discussion of `valley.c`

CS1010 Laboratory 08

Zhang Puyu

Exercise 5 Review

Read in Character Matrices

Frequently Seen Bugs

How to Solve Last 3???

C Pre-processing

Bonus Info

Searching And Sorting

Exercise 6

- **Questions** to test your understanding of strict V-arrays:
  - Let $v[k]$ to be the **minimal** element of a strict V-array of size n, if we know that $v[i-1] > v[i]$, what can we deduce about i and k?
  - How about $v[i+1] > v[i]$? What can we deduce about i and k then?
  - Can there exist $i \neq j$ such that $v[i] = v[j]$?
  - What if we add the condition that $i, j \leq k$? Can $v[i] = v[j]$ now?
- Now **discuss:**
  - What is an (obvious) $\mathcal{O}(n)$ solution?
  - What is an $\mathcal{O}(\log n)$ solution? (Hint: $\log n$ suggests binary search. Try to half the array and see what happens.)

Higher-Level Discussion of `inversion.c`

- The $\mathcal{O}\left(n^2\right)$ solution is just `kendall.c`.
- **Discuss**:
  - A small math challenge: without counting, how many inversions are there in the second half of the array (after the maximum) if that half contains n elements?
  - How to implement an $\mathcal{O}\left(n \log n\right)$ solution? (Hint: $n \log n$ suggests n times of binary search. The question is - what are we searching?)
  - How to implement an $\mathcal{O}(n)$ solution? (Hint: Do you realise that there is no inversion in the first half of the array?)

Higher-Level Discussion of `marks.c`

- If we only need to sort the scores, can it be easily done in $\mathcal{O}(n)$ time?
- Now, the only difference is that each score is associated with a name.
- So when the scores are moved around during the sorting, the indices of the names should move around accordingly.
- The problem is: how to maintain the ordering of the names during sorting?
- Observe: "lexicographical order" means that if we iterate the score-name pairs from top to bottom, the names are guaranteed to be sorted.
- Can we pre-compute the position for each score before starting moving the pairs around?