ENGR151 Recitation Class 6

week 8

Wu JiaXi

UM-SJTU joint institute

nina_nhk@sjtu.edu.cn

Computer & Programming, MatLab-scripting
November 18, 2024

Presentation Overview

1. Playbook Review
   C7
2. Worksheet Review
   W7
3. Referencing

Recording

Here is the link for the recording of this RC: Click Here

PRESENTATION BODY SLIDES

Disclaimer:
The answers provided here are not guranteed to be correct. Please use them as a references only and verify with reliable source.

Note:
only go through some questions that we think they are necessary.

C7 review 7.1.1

1. What is the first index in an array?
In C, the first index of an array is always 0.
2. Can we write `int a[] = {1, 2, 3, 4}`? Why?
Yes, this is valid because C allows the compiler to determine the array size based on the number of elements in the initializer.
3. Why no braces are used in lines 4-5 of the for loop?
If a loop body consists of only one statement, braces are optional.

C7 review 7.1.2

1. Why is `arr[]` used instead of `arr[5]`?
`arr[]` is used because the array size is passed as a separate parameter. It doesn't need to be specified in the function signature.
2. What is `const` and why is it used?
`const` makes the `size` parameter read-only within the function to prevent accidental modification.
3. What is `size_t` and why is it used?
`size_t` is an unsigned integer type used for array sizes and indexing, ensuring that the size can't be negative.

C7 review 7.1.4

1. Why is `srand(time(NULL))` used?
`srand(time(NULL))` seeds the random number generator with the current time to ensure a different sequence of random numbers each time the program runs.
2. What is `time.h`?
`time.h` is a library providing time and date functions in C.
3. How many times does the random seed need to be initialized?
The random seed should be initialized once at the start of the program.
4. Run the program and explain why the result is meaningful.
The result is meaningful because it simulates the distribution of dice rolls, showing how often each face of the die appears in a series of rolls.

C7 review 7.1.6

1. Why should multidimensional arrays be avoided?
Multidimensional arrays can be more difficult to manage and less memory-efficient.
2. Rewrite the code to use a 1-dimensional array.

```
int table[1000];
```

3. Increase the number of rolls by adding 0's. What happens?
Adding 0's could make the simulation behave differently depending on how the array is handled. It may result in invalid memory accesses.

C7 review 7.2.1

1. What are pointers?
A pointer is a variable that stores the memory address of another variable.
2. Are `int* a` and `int *a` the same?
Yes, both notations mean the same: `a` is a pointer to an integer.
3. Meaning of `*` in `*a = 5` and `int* a`?
In `*a = 5`, `*a` dereferences the pointer to assign a value. In `int* a`, `*` indicates that `a` is a pointer.
4. Double pointer and triple pointer illustration:
`int **p;` points to a pointer, and `int ***p;` points to a double pointer.

C7 review 7.2.5

1. What is `float **xp2` and why is it used?
`xp2` is a pointer to a pointer. It holds the address of a pointer variable.
2. Where is `xp1` pointing?
`xp1` is pointing to the variable `x`.
3. Why is the compiler issuing a warning?
The warning may be due to using the wrong format specifier for pointer types in `printf`.

C7 review 7.2.6

1. Why should the size of the element be specified in `malloc()`?
This ensures that the correct amount of memory is allocated for each element.
2. What is a memory leak?
A memory leak occurs when dynamically allocated memory is not freed, leading to wasted memory.
3. Which is safer to use in `engr151`, `malloc` or `calloc`?
`calloc` is safer because it initializes the allocated memory to zero.
4. Should memory be freed before using `realloc`?
Yes, the previous memory should be freed before reallocating.

C7 review 7.2.7

1. What is a segmentation fault and why does it happen?
A segmentation fault occurs when a program attempts to access invalid memory. It happens when pointers point to incorrect locations.
2. Why are pointers represented from right to left?
This notation is used to clarify that the asterisk (*) applies to the variable, not the type.

C7 review 7.2.8

1. How easy is it to use pointers with a structure?
Using pointers with structures allows for dynamic memory allocation but can be complex if not handled properly.
2. What is the size of `person_t`?
The size of `person_t` depends on the system architecture and padding, but it includes the size of each field.
3. What is the benefit of defining a structure variable?
Defining a variable like `ya` allows for easy initialization and access to members.
4. What is the difference between `.` and `->`?
`.` is used when accessing structure members directly, and `->` is used when accessing members via a pointer.

C7 review 7.2.10

Why is it Impossible to Request a Specific Address?

- Memory addresses are managed by the operating system, and it controls memory allocation.
- Programs cannot directly request specific memory addresses because they may not be available or reserved.

Why Check if malloc() Returns NULL?

- `malloc()` can fail to allocate memory, returning a NULL pointer.
- Dereferencing a NULL pointer leads to undefined behavior, typically a crash (segmentation fault).

C7 review 7.2.10

Why Initialize Pointers to NULL?

- Uninitialized pointers point to random memory, causing undefined behavior.
- Initializing to `NULL` ensures safe pointer operations, as dereferencing a NULL pointer is easily detectable.

C7 review 7.3.1

Uncomment Instructions and Explain Behavior

- Uncommenting lines 12 and 14 alters the pointer assignments.
- When doing `p = a`, `p` points to the same location as `a`.
- But when `a = p` is attempted, `a` cannot be reassigned, as it's a local array.

Why Can You Do `p = a` But Not `a = p`?

- `p` is a pointer and can point to any memory location.
- `a` is a fixed array with a predefined memory location, so it cannot be reassigned like `p`.

C7 review 7.3.1

The Problem with `p = a`

- Before assignment, `pp` (or `p`) may have pointed to an address that no longer holds valid memory after the reassignment.

Is `p++` Good?

- `p++` increments the pointer, but this can be dangerous if `p` moves beyond allocated memory or changes the intended structure.
- It is only useful when you need to traverse an array or memory block.

C7 review 7.3.2

Is Everything Valid for `char*`?

- Yes, the same principles apply for `char*` as for other types of pointers.

Why Can't We Change `*p`?

- If `p` points to a constant memory location, changing `*p` will result in undefined behavior.
- Modifying a constant pointer is not allowed.

Why Don't We Use `malloc()` or `calloc()` for `p`?

- `p` is directly initialized to a predefined block of memory and doesn't require dynamic allocation.

C7 review 7.3.3

Why Is `p` Freed on Line 6?

- `p` is dynamically allocated with `calloc()` and should be freed after use to prevent memory leaks.

Can a Pointer be Accessed as an Array?

- Yes, `p[2]` is equivalent to `*(p+2)`.
- Pointers and arrays are closely related in C, and array indexing is just pointer arithmetic.

C7 review 7.3.5

Why is `table` Not Multi-Dimensional Anymore?

- The array is allocated linearly, and using `malloc()` or `calloc()` to allocate memory dynamically removes the static 2D structure.

How Are `malloc()` and `calloc()` Used?

- `malloc()` allocates memory without initializing it, while `calloc()` allocates and zeroes the memory.
- `calloc()` is safer when initializing arrays, preventing garbage values.

How Can We Increase Rolls Without Crashing?

- By dynamically allocating memory with `malloc()` or `calloc()`, we ensure that memory is allocated as needed.
- Increasing the number of rolls doesn't cause stack overflow or memory crashes.

C7 review 7.3.7

Explanation of `arr_as_ptr.c`

- The code demonstrates pointer manipulation and memory allocation with `calloc()`.
- `p[0] = 1;` assigns to the dynamically allocated array, while `*(p + 2) = 3;` modifies another element.
- The code prints the values of both the statically allocated array `a[]` and the dynamically allocated `p[]`.

How to Get More Than One Output?

- You can call multiple print statements or modify the array and print the results after each modification.
- The output comes from each print statement inside loops or function calls.

C7 review 7.3.7

Memory Errors and Crashes

- Dereferencing NULL pointers.
- Accessing memory out of bounds.
- Failing to free dynamically allocated memory (memory leaks).
- Double freeing memory (double free error).

Note: For some questions, we won't directly provide the entire source code. Instead, we will provide ideas, a piece of code or pseudo-code to help guide you through worksheet questions.

W7 Review

Exercise 2: Given 10 user-inputted marks, calculate:

- The mean
- The median
- The variance
- The standard deviation

Each calculation should be done in a separate function, and results stored in an array.

Solution for Mean

Mean Calculation:

$$\text{Mean} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

where x_i are the marks, and $N = 10$.

`meanCalculation` function:

- Takes an array of marks.
- Returns the mean.

Solution for Median

Median Calculation:

$$\text{Median} = \begin{cases} x_{\frac{N}{2}} & \text{if } N \text{ is odd} \\ \frac{x_{\frac{N}{2}} + x_{\frac{N}{2}+1}}{2} & \text{if } N \text{ is even} \end{cases}$$

`medianCalculation` function:

- Sorts the array.
- Returns the median.

Solution for Variance and Standard Deviation

Variance Calculation:

$$\text{Variance} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \text{Mean})^2$$

Standard Deviation:

$$\text{Std Dev} = \sqrt{\text{Variance}}$$

`varianceCalculation` and `stdDevCalculation` functions:

- `varianceCalculation` calculates the variance.
- `stdDevCalculation` calculates the standard deviation.

Exercise 3

1. Create a 1000×1000 matrix A with random integers.
2. Display all elements of A in reverse order.
3. Read elements of A row-wise and column-wise.
4. Compare the time taken for row-major vs. column-major access.

Solution to Matrix Operations

1. Create Matrix:

$$A = \text{Random integers } (1000 \times 1000)$$

2. Display Reverse:

- Traverse A in reverse order.

3. Row-major and Column-major Read:

- Row-major: Traverse each row in sequence.
- Column-major: Traverse each column in sequence.

Exercise 4

Given integers m and n, a pointer p to an integer, and a pointer pp to a pointer to an integer, perform these operations:

1. Initialize $m = 5$, $n = 2$; display addresses of m and n.
2. Set p to point to m and display p and value at p.
3. Set pp to point to p; display pp and value at pp.
4. Set p to point to n; display p, pp, and values.
5. Set p to m and pp to n; explain results.

Pointer Operations Explained

1. **Address Initialization:**
   - $m = 5$, $n = 2$
   - Print addresses: `&m` and `&n`
2. **Pointer to m:**
   - p points to m
   - $*p = 5$ (value at address)

Pointer-to-Pointer Explanation

3. Pointer to Pointer (pp):

- pp points to p
- $*pp = p$; $**pp = 5$ (value of m)

Explanation: pp holds the address of p, and $*pp$ dereferences p to get m's value.

Pointer Reassignment

4. Reassign p to n:

- p now points to n with value $*p = 2$
- pp remains unchanged, still pointing to original p

5. Set $p = m$, $pp = n$:

- Illegal operation since pp cannot point to a non-pointer type like n.

Exercise 5

Exercise 5

Ex. 5 — *Guess what!*

1. Given the following sample code, which of the following statements will change the value of m? If there are invalid statements explain the problem.

   a) `n=1773;`
   b) `*n=4567;`
   c) `p=m++;`
   d) `*p=n++;`

```c
int *p; int m, n;
m = 42; n = m; p = &m;
```

Exercise 5

2. A student wrote the following sample code.

   a) Fix anything that needs to be fixed in the code.

   b) Assuming $c1$ and $c2$ are stored at 12345 and 67890, respectively, what will be displayed?

```
1 char c1 = 'G', c2 = 'M';
2 char *p1 = c1, *p2 = &c2, *p3;
3 printf("%p %p %p\n", p3, p2, p1);
4 p3 = &c2; printf("%c %p",p3, *p3)
5 p3 = p1; printf("%p %c", p3, *p3);
6 *p1 = *p2; printf("%p %c", p1, *p1)
```

3. A student, who was too tired to focus during the lecture, wrote the following code.

   a) Will the compiler issue a warning or an error? What message will be displayed?

   b) Explain to the student what is wrong with his code and provid a hint to fix it.

```
1 char c = 'G';
2 double *p = &c;
```

Note p is a pointer for double

Exercise 6

Exercise 6

Ex. 6 — *I hate pointers!*

After a few sleepless night fighting with pointers a poor ENGR151 student feels totally lost. He comes to you in full despair with the following program. What will you do?

```
1  #include<stdio.h>
2  int main() {
3    int b, *c;
4    c=test(b);      ⟶ data type of C
5    free(c);
6  }
7  int test(int *b) {
8    int a; int** c; int*** d;
9    a=4; b=&a; *b<<=1; a++;
10   b=malloc(2*sizeof(int));   ⟶ the address of a is lost.
11   *(b+1)=a; *(b+1)<<=2;
12   c=&b; d=&c; **c+=1<<3;
13   printf("%d %d\n",*(b+1),***d);
14   return c;          ⟶ return type doesn't match
15   free(b);           ⟶ memory isn't free
16 }
```

References

- Manuel. *c5.pdf*. JI Canvas, 2024. [Link].
- Manuel. *w5.pdf*. JI Canvas, 2024. [Link].

The End

Questions? Comments?