INFO1111: Computing 1A Professionalism

2023 Semester 1

Self-Learning Report

Submission number: 1

Github link: https://github.com/YuriG5307/info1111.git

Student name	Yiyang Gao
Student ID	510015154
Topic	C *Note: This must be the same as was in your topic approval*
Levels already achieved	Level A, B, C
Levels in this report	Level A, B, C

Contents

Instructions

Important: This section should be removed prior to submission.

You should use this LaTeX template to generate your self-learning report. Keep in mind the following key points:

- **Submissions**: There will be three opportunities during the semester to submit this report. For each submission you can attempt 1 or 2 levels. Each submission should use the same report, but amended to include new information.
- **Assessment**: In order to achieve level B, you must first have achieved level A, and so on for each level up to level D. This means that we will not assess a higher level until a lower level has been achieved (though we will review one level higher and give you feedback to help you in refining your work).
- **Minimum requirement**: Remember that in order to pass the unit, you must achieve at least level A in the self-learning (unless you achieve level B in both the skills and knowledge categories).
- **Using this template**: When completing each section you should remove the explanation text and replace it with your material.
- **Referencing**: You should also ensure that any resources you use are suitably referenced, and references are included into the reference list at the end of this document. You should use the IEEE reference style [1] (the reference included here shows you how this can be easily achieved).

1. Level A: Initial Understanding

1.1. Level A Demonstration

List the three things you will do to demonstrate your understanding of this topic. *Note: This must be the same as was in your topic approval*

Compilation, linking, use of libraries

data types and operators

pointers, string manipulation

1.2. Learning Approach

How did you approach your learning? Write 100 - 200 words outlining the steps you took and/or are taking to self-learn the topic you have selected.

During my self-teaching of C language, I first built up basic concepts and theoretical knowledge by reading classic C language tutorials and online resources. These textbooks helped me understand the basic concepts of the C language, such as the basic structure, data types, control structures, functions, and pointers. At the same time, I also referred to some high-quality online courses.

After establishing the basic theoretical knowledge, I started to practice and try to write some simple C language programs. I deepen my understanding of C language knowledge by solving practical problems.

In order to consolidate the knowledge I have learned and learn more advanced C language skills, I often communicate with my friends, share experiences, and learn from each other. At the same time, record my learning process and content to help consolidate knowledge and improve my ability to write code.

In short, in the process of self-learning C language, I used a variety of learning resources, combined theoretical learning and practical operation, and continuously improved my skills through discussions with friends. This comprehensive learning method has enabled me to master the C language in a relatively short period of time.

1.3. Challenges and Difficulties

What did you find most difficult? Write 100 - 200 words discussing what you have or are finding most challenging about self-learning the topic you have selected (e.g. are there any elements of the topic you have found more difficult to learn than others etc.).

In the process of teaching myself C language, I found that the most challenging part is to understand pointer manipulation techniques in depth. These concepts are more unique and difficult to understand than other programming basics, so I put more energy and time into research.

As a unique concept in C language, pointer involves address reference and memory space operation. For beginners like me, these contents may be difficult to understand. In

order to master the usage of pointers, I tried to use them many times in actual programming, and gradually understood how to use pointers in dynamic memory allocation, array operations, and function parameter passing.

Although I encountered many challenges in the process of self-learning C language, I gradually overcome these difficulties through continuous practice, experience accumulation and interaction with others. These challenges not only gave me a deeper understanding of the underlying principles of the C language, improved my programming skills, but also inspired my enthusiasm to continue exploring other advanced technologies.

1.4. Learning Sources

Learning Source - What source did you use? (Note: Include source details such as links to websites, videos etc.). Contribution to Learning - How did the source contribute to your learning (i.e. what did you use the source for)?

Learning Source - What source did you use? (Note: Include source details such as links to websites, videos etc.).	Contribution to Learning - How did the source contribute to your learning (i.e. what did you use the source for)?
https://tinyurl.com/2tkhten6	Linking and Libraries
https://tinyurl.com/55fyfcn6	Compilation
https://tinyurl.com/bdzj47c9	Operators
https://tinyurl.com/3v4b26zb	Types
https://tinyurl.com/58fe7c3w	Pointers, String Manipulation

1.5. Application artifacts

Include here a description of what you actually created (what does it do? How does it work? How did you create it?). Include any code or other related artefacts that you created (these should also be included in your github repository).

If you do include screengrabs to show what you have done then these should be annotated to explain what it is showing and what the application does.

fetch_url_content.c

```c
#include <stdio.h>
#include <curl/curl.h>

size_t write_callback(void *content, size_t size, size_t nmemb, void *userp) {
    fwrite(content, size, nmemb, (FILE *)userp);
    return size * nmemb;
}

int main() {
    CURL *curl_handle;
    CURLcode res;

    curl_global_init(CURL_GLOBAL_DEFAULT);
    curl_handle = curl_easy_init();
    if (curl_handle) {
        curl_easy_setopt(curl_handle, CURLOPT_URL, "https://www.1234567.com");
        curl_easy_setopt(curl_handle, CURLOPT_WRITEFUNCTION, write_callback);

        FILE *file = fopen("output.html", "wb");
        if (file) {
            curl_easy_setopt(curl_handle, CURLOPT_WRITEDATA, file);
            res = curl_easy_perform(curl_handle);
            fclose(file);

            if (res != CURLE_OK) {
                fprintf(stderr, "curl_easy_perform() failed: %s\n", curl_easy_strerror(res));
            } else {
                printf("URL content saved to output.html\n");
            }
        } else {
            fprintf(stderr, "Unable to create output file.\n");
        }

        curl_easy_cleanup(curl_handle);
    }

    curl_global_cleanup();
    return 0;
}
```

This program uses the libcurl library to send a simple GET request to the specified URL and saves the content to a file named output.html. Next, you need to compile and link the program (you can type in the terminal: gcc -o fetch.url.content fetch.url.content.c -lcurl)

which will compile the fetch.url.content.c source file and link to the libcurl library. The -o option specifies the output executable file name, and the -lcurl option indicates linking to the libcurl library. After compiling, you will get an executable named fetch.url.content. You can then enter in the terminal (./fetch.url.content)

At this time, the program will send a GET request to https://www.1234567.com, and save the response content to the output.html file. If the request is successful, it will output "URL content saved to output.html".

types and operators.c

```c
#include <stdio.h>

int main() {
    // Declare different data types
    int a = 1, b = 2;
    float c = 1.1, d = 2.2;
    char e = 'A';

    // Use integer operators
    printf("Integer operations results:\n");
    printf("a + b = %d\n", a + b);
    printf("a - b = %d\n", a - b);
    printf("a * b = %d\n", a * b);
    printf("a / b = %d\n", a / b);
    printf("a %% b = %d\n\n", a % b);

    // Use floating point operators
    printf("Floating-point operations results:\n");
    printf("c + d = %.2f\n", c + d);
    printf("c - d = %.2f\n", c - d);
    printf("c * d = %.2f\n", c * d);
    printf("c / d = %.2f\n\n", c / d);

    // Use relational operators
    printf("Relational operations results:\n");
    printf("a > b: %s\n", a > b ? "true" : "false");
    printf("a < b: %s\n", a < b ? "true" : "false");
    printf("a == b: %s\n", a == b ? "true" : "false");
    printf("a != b: %s\n\n", a != b ? "true" : "false");

    // use logical operators
    printf("Logical operations results:\n");
    int result = (a > b) && (c < d);
    printf("(a > b) && (c < d): %s\n", result ? "true" : "false");
    result = (a > b) || (c < d);
    printf("(a > b) || (c < d): %s\n\n", result ? "true" : "false");

    // use bitwise operators
    printf("Bitwise operations results:\n");
    printf("a & b = %d\n", a & b);
    printf("a | b = %d\n", a | b);
    printf("a ^ b = %d\n", a ^ b);
    printf("~a = %d\n", ~a);
    printf("a << 1 = %d\n", a << 1);
    printf("a >> 1 = %d\n\n", a >> 1);

    // use character operators
    printf("Character operations results:\n");
    printf("Lowercase of uppercase letter %c is %c\n", e, e + 32);

    return 0;
}
```

This program demonstrates the basic data types (integer, floating point, character) and various operators (arithmetic, relational, logical, bit, character) in C language. By running this program, that can observe the usage and results of different operators.

pointers, string manipulation.c

```c
#include <stdio.h>
#include <string.h>

void reverse_string(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = temp;
    }
}

int main() {
    char my_string[] = "Hello, world!";
    printf("Original string: %s\n", my_string);

    reverse_string(my_string);
    printf("Reversed string: %s\n", my_string);

    return 0;
}
```

This program defines a function that reverses an incoming string. We use a pointer as an argument, through which the original string is directly modified. In the main function, we create a new string and call the function defined at the beginning to reverse it. Then, we print the original string and the reversed string.

2. Level B: Basic Application

Whilst level A is about doing something simple with the topic to just show that you have started to be able to use the tool or technology, level B is about doing something practical that might actually be useful.

2.1. Level B Demonstration

This is a short description of the application that you have developed in order to demonstration your understanding. (50-100 words).

I have developed an application in C language that integrates the basic concepts of C language that I have learned, such as basic data types, pointers, and the use of libraries. This application is a simple number guessing game, where players are prompted to guess a random number between 1 and 100. Through this game, I can practice and demonstrate my understanding and application of the basic concepts of C language.

2.2. Application artifacts

Include here a description of what you actually created (what does it do? How does it work? How did you create it?). Include any code or other related artefacts that you created (these should also be included in your github repository).

If you do include screengrabs to show what you have done then these should be annotated to explain what it is showing and what the application does.

The number guessing game is an interactive console application. When the program starts, it generates a random number between 1 and 100. Then, the program prompts the player to input their guess, and the program gives feedback based on whether the player's guessed number is too high, too low, or correct, until the player guesses correctly.

In the process of creating this program, I used various basic data types in C language, such as integers (for storing the secret number and the player's guess) and characters (for receiving player's input). At the same time, I also used pointers to dynamically get player's input. In addition, I utilized the standard library functions in C language, such as rand() to generate random numbers, and printf(), scanf() etc. to interact with the player.

Here's the code for the game:

guessing.game.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    int secretNumber, userGuess;

    // Initialize the random number generator with the current time
    srand(time(NULL));

    // Generate random numbers from 1 to 100
    secretNumber = rand() % 100 + 1;

    printf("Welcome to the number guessing game!\n");
    printf("I'm thinking of a number between 1 and 100.\n");

    do {
        printf("Enter your guess: ");
        scanf("%d", &userGuess); // Read the player's guesses

        if (userGuess > secretNumber) {
            printf("Too high! Try again.\n");
        } else if (userGuess < secretNumber) {
            printf("Too low! Try again.\n");
        }
    } while (userGuess != secretNumber);

    printf("Congratulations! You've guessed the number.\n");

    return 0;
}
```

To compile and run this program, you can save this code as a guessing.game.c file, and then enter the following command in the command line:

gcc -o guessing.game guessing.game.c
./guessing.game

This will compile and run the number guessing game. This application is a practical interactive program that helps me practice and showcase my understanding and application of the C language.

3. Level C: Deeper Understanding

Level C focuses on showing that you have actually understood the tool or technology at a relatively advanced level. You will need to compare it to alternatives, identifying key strengths and weaknesses, and the areas where this tool is most effective.

3.1. Strengths

What are the key strengths of the item you have learnt? (50-100 words)

C language allows direct manipulation of computer memory and hardware. This is extremely useful when you need to write programs that require high speed or need to communicate directly with hardware. In addition, many popular programming languages today, such as C++ and Java, are based on C language, so learning C makes understanding these languages much easier.

3.2. Weaknesses

What are the key weaknesses of the item you have learnt? (50-100 words)

However, C language also has some weaknesses. For example, it does not manage memory automatically like some newer programming languages, so programmers need to manage it themselves. If not managed well, it can lead to errors or security issues. Moreover, the syntax of C language is relatively complex, which is not very friendly to beginners.

3.3. Usefulness

Describe one scenario under which you believe the topic you have learnt could be useful. (50-100 words)

C language is useful in many places. For example, it is often used to write programs for embedded systems. Embedded systems are like the televisions, microwaves, and printers we use daily. Their hardware resources are limited, and they need a programming language that can directly control the hardware and run efficiently. C language can do this.

3.4. Key Question 1

Note: This question is in the table in the 'Self Learning: List of Topics' page on Canvas. (50-100 words)

If need to write a program that interacts directly with hardware or needs fine control over memory usage, then C language would be better than Python. In addition, if your program needs to perform a large amount of mathematical calculations or graphics processing, and speed is important, then C language will usually run faster than Python.

3.5. Key Question 2

Note: This question is in the table in the 'Self Learning: List of Topics' page on Canvas. (50-100 words)

In C language, we generally divide the program into header files and source files. Header files are where some function declarations, macro definitions, and type definitions

are stored. They are usually shared by multiple source files. Source files contain the actual implementation of functions, which the compiler converts into code that the machine can understand. Source files include header files, allowing the source files to use the functions and types declared in the header files.

4. Level D: Evolution of skills

4.1. Level D Demonstration

This is a short description of the application that you have developed. (50-100 words). **IMPORTANT:** *You might wish to submit this as part of an earlier submission in order to obtain feedback as to whether this is likely to be acceptable for level D.*

4.2. Application artifacts

Include here a description of what you actually created (what does it do? How does it work? How did you create it?). Include any code or other related artefacts that you created (these should also be included in your github repository).

If you do include screengrabs to show what you have done then these should be annotated to explain what it is showing and what the application does.

4.3. Alternative tools/technologies

Identify 2 alternative tools/technologies that can be used instead of the one you studied for your topic. (e.g. if your topic was Python, then you might identify Java and Golang)

4.4. Comparative Analysis

Describe situations in which both your topic and each of the identified alternatives would be preferred over the others (100-200 words).

Bibliography

[1] The University of Sydney, "Referencing and citation styles: IEEE," 2022, see https://libguides.library.usyd.edu.au/c.php?g=508212.