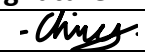
UNIVERSITI MALAYSIA SARAWAK

Faculty of Computer Science and Information Technology

Student Name	Student Id Number	Signature
Christopher Sii How Chiong	69385	

Subject Code: **TMN4133**	Subject Name: **System Programming**
Assignment Title: **Assignment**	Lecturer : **Associate Professor Dr. Johari bin Abdullah**
Due Date: **25th November 2022**	Date Submitted: 21st November 2022

Plagiarism and Collusion are methods of cheating that falls under Peraturan Akademik Universiti Malaysia Sarawak para 11: Etika Akademik

Plagiarism

Plagiarism is the presentation of work which has been copied in whole or in part from another person's work, or from any other source such as the internet, published books or periodicals without due acknowledgement given in the text.

Collusion

Collusion is the presentation of work that is the result in whole or in part of unauthorized collaboration with another person or persons.

Where there are reasonable grounds for believing that cheating has occurred, the only action that may be taken when plagiarism or collusion is detected is for the staff member not to mark the item of work and to report or refer the matter to the Dean. This may result in work being disallowed and given a fail grade or if the circumstances warrant, the matter may be referred to a Committee of inquiry for investigation. Such investigation may result in the matter being referred to the University Discipline Committee, **which** has the power to exclude a student.

Upon placing signature above, I certify that I have not plagiarized the work of others or participated in unauthorized collusion when preparing this assignment.

I also certify that I have taken proper case in safeguarding my work and have made all reasonable efforts to ensure that may work not be able to be copied.

MARK :

Table of Contents

List of Tables

List of Figures

1.0 Introduction about Github Copilot

GitHub Copilot is an artificial intelligence-powered pair programmer that provides autocomplete-style recommendations as users code. Users can get recommendations from GitHub Copilot by either beginning to write the code users want to use or by leaving a textual post detailing what users intend the code to do. GitHub Copilot analyses the context in the file users are editing as well as related files and makes recommendations from within the user's text editor. OpenAI Codex, a new AI system developed by OpenAI, powers GitHub Copilot. GitHub Copilot is an extension for Visual Studio Code, Visual Studio, Neovim, and the JetBrains IDE suite.

GitHub Copilot has been trained on all languages available in public repositories. The volume and diversity of training data for each language may influence the quality of suggestions users receive. JavaScript, for example, is well-represented in public repositories and is one of the best supported languages on GitHub Copilot. Languages with fewer suggestions in public repositories may produce fewer or less robust suggestions.

GitHub Copilot makes recommendations based on a model built by OpenAI from billions of lines of open source code. As a result, the GitHub Copilot training set may contain insecure coding patterns, bugs, or references to obsolete APIs or idioms. When GitHub Copilot generates recommendations based on this training data, those recommendations may contain undesirable patterns.

Users are accountable for the security and quality of their code. It is recommended that users take the same precautions when using GitHub Copilot code as they would when using any code that users did not write themselves. These precautions include rigorous testing, IP scanning, and vulnerability tracking. GitHub Actions, Dependabot, CodeQL, and code scanning are some of the tools available to help programmer monitor and improve code quality. All of these features are available for free in public repositories.

According to the GitHub Copilot website, GitHub Copilot includes programmers-friendly features such as code comment conversion to runnable code and autocomplete for chunks of code, repetitive sections of code, and entire methods and/or functions. According to GitHub, Copilot's autocomplete feature is roughly half accurate; for example, with some Python function header code, Copilot correctly autocompleted the rest of the function body code 43% of the time on the first try and 57% of the time after ten attempts. The features of Copilot help programmers navigate unfamiliar coding frameworks and languages by reducing the amount of time users spend reading documentation.

2.0 Screenshots for proof of registration for Github and Copilot

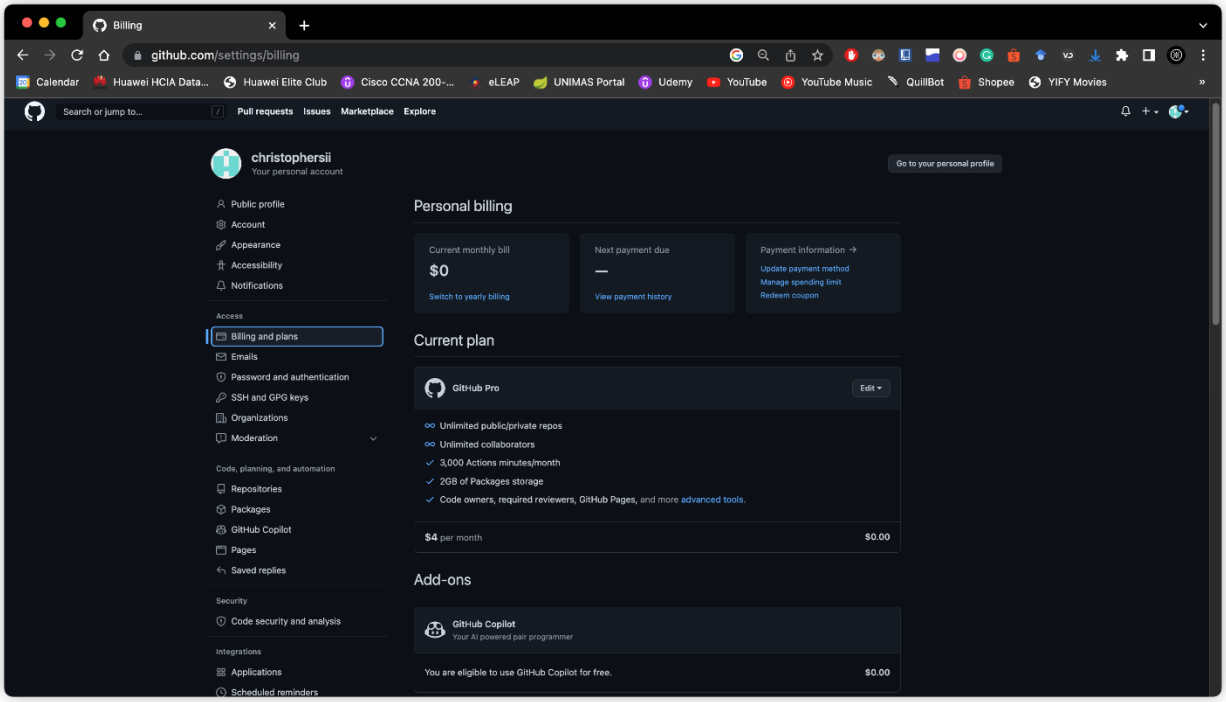


Figure 1 registration for Github Copilot using siswa account.

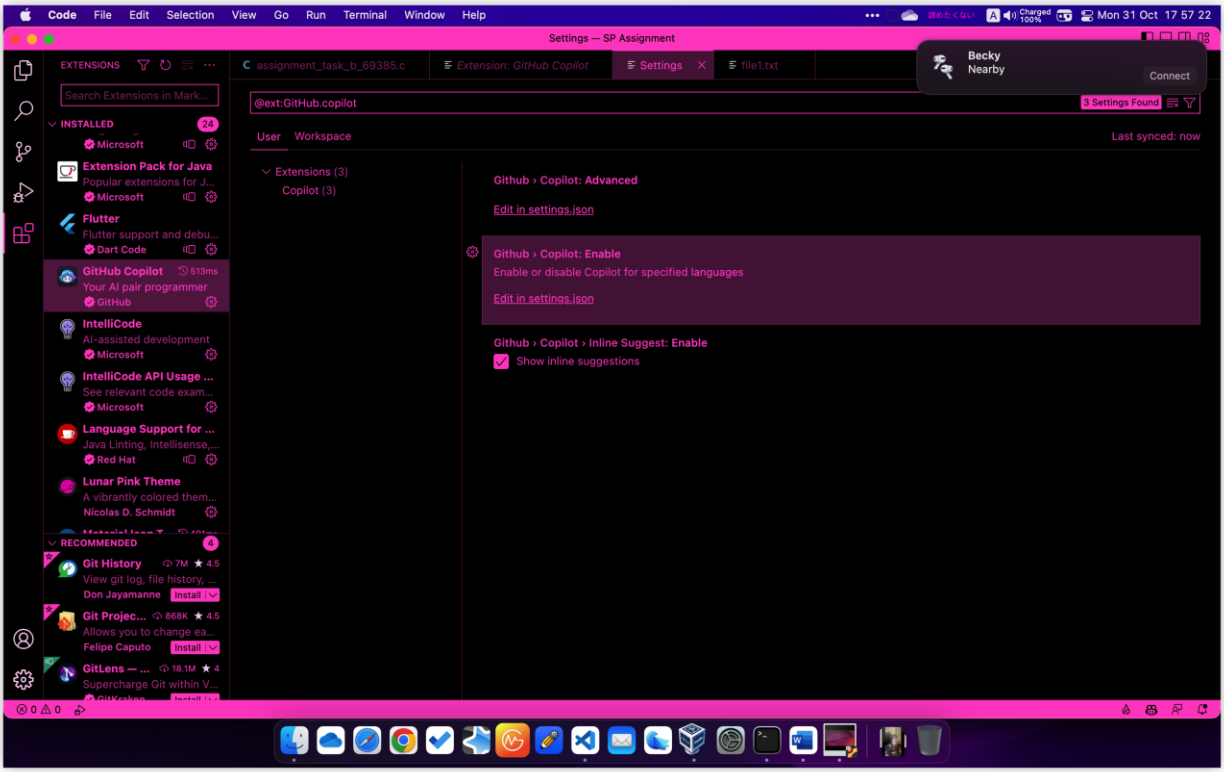


Figure 2 Visual Studio Code Github Copilot setting page.

3.0 Task A

Program in C to read n number of values in an array and display it in reverse order.

3.1 Screenshot

```
chris@chris-VirtualBox:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
chris@chris-VirtualBox:~$ cd Desktop
chris@chris-VirtualBox:~/Desktop$ ls
chris@chris-VirtualBox:~/Desktop$ mkdir sp_assignment
chris@chris-VirtualBox:~/Desktop$ ls
sp_assignment
chris@chris-VirtualBox:~/Desktop$ cd sp_assignment
chris@chris-VirtualBox:~/Desktop/sp_assignment$ ls
chris@chris-VirtualBox:~/Desktop/sp_assignment$ nano task_a
chris@chris-VirtualBox:~/Desktop/sp_assignment$ ls
task_a  task_a.c
chris@chris-VirtualBox:~/Desktop/sp_assignment$ rm task_a
chris@chris-VirtualBox:~/Desktop/sp_assignment$ ls
task_a.c
chris@chris-VirtualBox:~/Desktop/sp_assignment$ gcc task_a.c -o task_a.o
task_a.c:1:10: fatal error: stdio.h: No such file or directory
    1 | #include <stdio.h>
      |          ^~~~~~~~~
compilation terminated.
chris@chris-VirtualBox:~/Desktop/sp_assignment$ nano task_a.c
chris@chris-VirtualBox:~/Desktop/sp_assignment$ gcc task_a.c -o task_a.o
task_a.c:1:9: fatal error: stdio.h: No such file or directory
    1 | #include<stdio.h>
      |         ^~~~~~~~~
compilation terminated.
chris@chris-VirtualBox:~/Desktop/sp_assignment$ ^C
chris@chris-VirtualBox:~/Desktop/sp_assignment$ sudo apt install libc6-dev
[sudo] password for chris:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

```
chris@chris-VirtualBox:~/Desktop/sp_assignment$ gcc task_a.c -o task_a.o
task_a.c: In function ‘main’:
task_a.c:23:35: error: expected ‘)’ before ‘;’ token
   23 |                 printf("% 5d",j[i];
      |                                   ^
      |                                   )
task_a.c:23:36: error: expected ‘;’ before ‘}’ token
   23 |                 printf("% 5d",j[i];
      |                                    ^
      |                                    ;
   24 |         }
      |         ~
chris@chris-VirtualBox:~/Desktop/sp_assignment$ nano task_a.c
chris@chris-VirtualBox:~/Desktop/sp_assignment$ gcc task_a.c -o task_a.o
chris@chris-VirtualBox:~/Desktop/sp_assignment$ ls
task_a.c  task_a.o
chris@chris-VirtualBox:~/Desktop/sp_assignment$ ./task_a.o


Read n number of values in an array and display it in reverse order:

Please enter the number of elements to store in the array: 3
Please enter 3 number of elements in the array:
Element- 0: 3
Element- 1: 2
Element- 2: 1

The values store into the array are:
    3    2    1

The values store into the array in reverse are:
    1    2    3
```

Figure 3 use of gcc compiler and nano editor for the manual approach and error encountered.

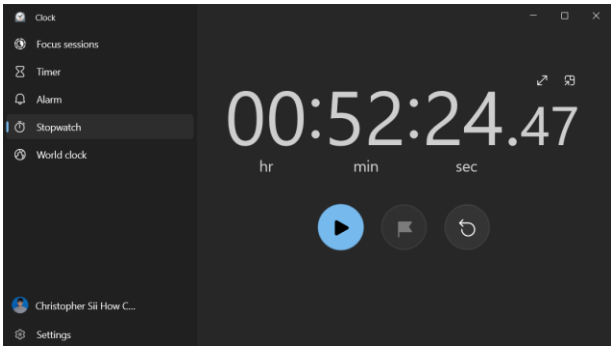


Figure 4 Time taken to complete task A for the manual approach.

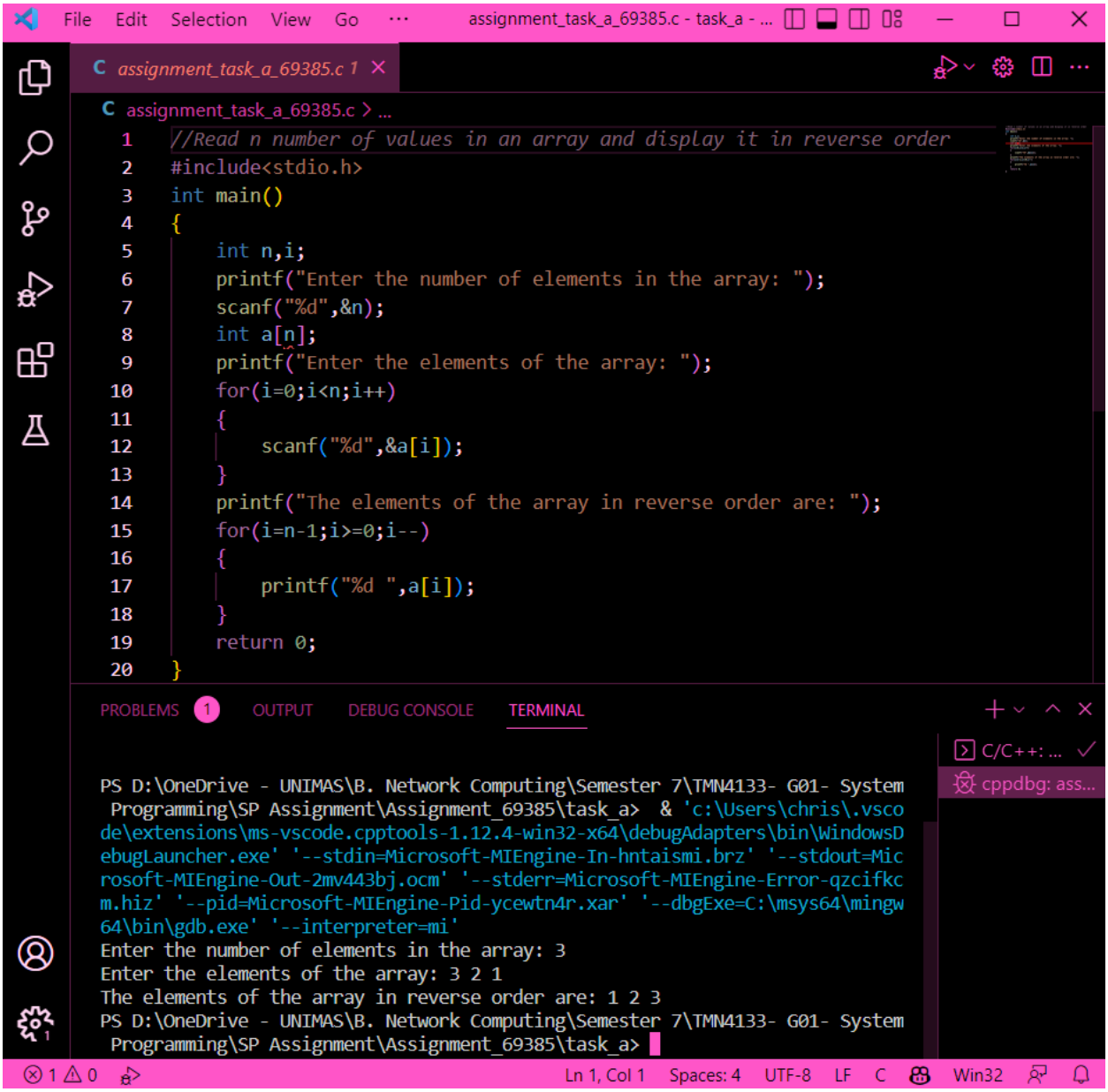


Figure 5 use of Visual Studio Code for the Github Copilot approach

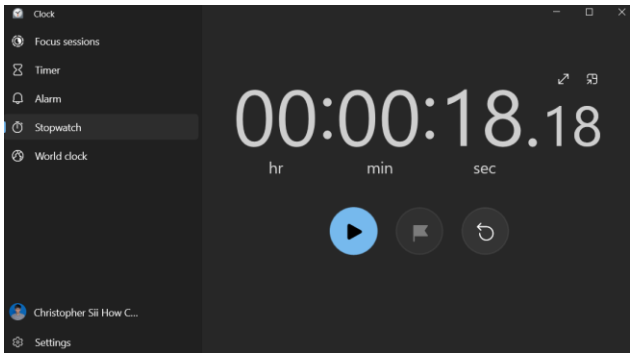


Figure 6 Time taken to complete task A for the VS Code and Github Copilot approach.

3.2 Parameters observed

Table 1 Parameter observation

	Manual approach	**VS Code and Github Copilot**
Time taken	52 minutes 24 seconds	18 seconds
Number of error(s)	2 errors	0 error

3.3 Code for manual and Copilot approach

```c
#include<stdio.h>

void main(){
    int i,n,j[100];

    printf("\n\nRead n number of values in an array and display it in reverse order:\n\n");
    printf("Please enter the number of elements to store in the array: ");
    scanf("%d", &n);

    printf("Please enter %d number of elements in the array: \n", n);
    for(i=0; i<n; i++){
        printf("Element- %d: " ,i);
        scanf("%d", &j[i]);
    }

    printf("\nThe values store into the array are: \n");
    for(i=0; i<n; i++){
        printf("% 5d", j[i]);
    }

    printf("\n\nThe values store into the array in reverse are:\n");
    for(i=n-1; i>=0; i--){
        printf("% 5d",j[i]);
    }

    printf("\n\n");
}
```

Figure 7 Code for manual approach.

```c
//Read n number of values in an array and display it in reverse order
#include<stdio.h>
int main()
{
    int n,i;
    printf("Enter the number of elements in the array: ");
    scanf("%d",&n);
    int a[n];
    printf("Enter the elements of the array: ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("The elements of the array in reverse order are: ");
    for(i=n-1;i>=0;i--)
    {
        printf("%d ",a[i]);
    }
    return 0;
}
```

Figure 8 Code for Copilot approach.

4.0 Task B

C program that enable to copy the content of one text file to another text file.

4.1 Screenshot

```
chris@chris-VirtualBox:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
chris@chris-VirtualBox:~$ cd Desktop
chris@chris-VirtualBox:~/Desktop$ ls
sp_assignment
chris@chris-VirtualBox:~/Desktop$ cd sp_assignment
chris@chris-VirtualBox:~/Desktop/sp_assignment$ ls
task_a.c  task_a.o
chris@chris-VirtualBox:~/Desktop/sp_assignment$ mkdir task_b
chris@chris-VirtualBox:~/Desktop/sp_assignment$ ls
task_a.c  task_a.o  task_b
chris@chris-VirtualBox:~/Desktop/sp_assignment$ cd task_b
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ ls
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ nano task_b.c
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ ls
task_b.c
```

```
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ ls
task_b.c  task_b.o
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ nano filename1.txt
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ nano filename2.txt
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ cat
^C
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ cat filename1.txt
HELLO WORLD! :D
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ cat filename2.txt
idk
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$
```

```
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ gcc task_b.c -o task_b.o
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ ./task_b.o
Enter the filename to open for reading
filename1.txt
Enter the filename to open for writing
filename2.txt

Contents copied to filename2.txtchris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ ls
filename1.txt  filename2.txt  task_b.c  task_b.o
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ cat filename1.txt
HELLO WORLD! :D
chris@chris-VirtualBox:~/Desktop/sp_assignment/task_b$ cat filename2.txt
HELLO WORLD! :D
```

Figure 9 use of gcc compiler and nano editor for the manual approach and error encountered.

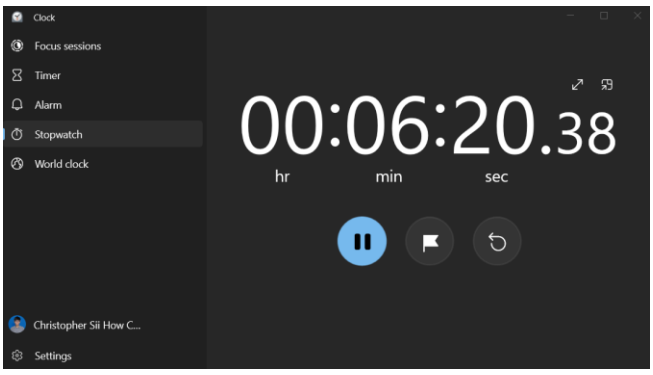


Figure 10 Time taken to complete task B for the manual approach

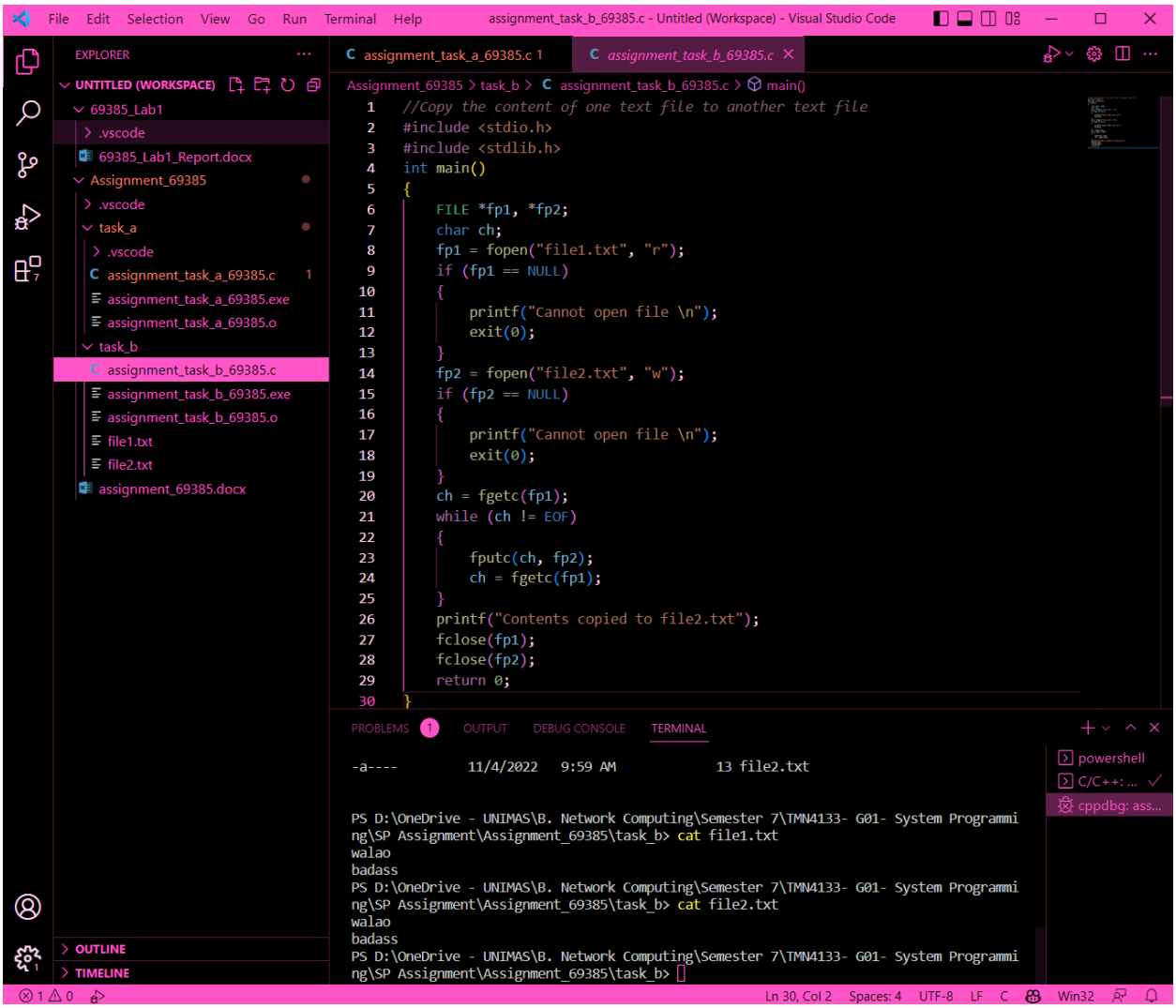


Figure 11 use of gcc compiler and nano editor for the copilot approach.

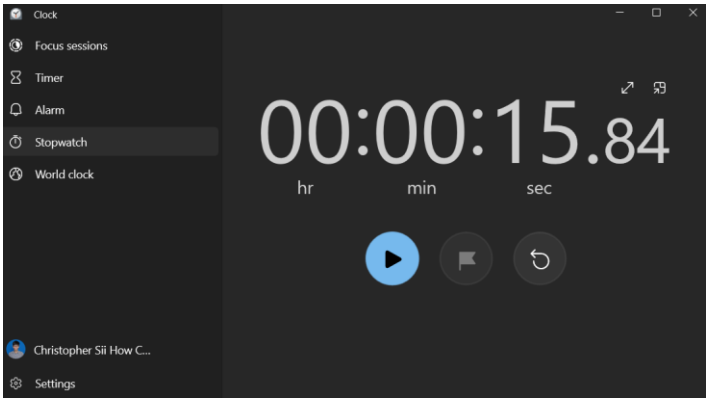


Figure 12 Time taken to complete task B for the copilot approach.

4.2 Parameters observed

Table 2 Parameter observation

	Manual approach	**VS Code and Github Copilot**
Time taken	06 minutes 20 seconds	15 seconds
Number of error(s)	0 error	0 error

4.3 Code for manual and Copilot approach

```c
#include <stdio.h>
#include <stdlib.h> // For exit()

int main() {
    FILE *fptr1, *fptr2;
    char filename[100], c;

    printf("Enter the filename to open for reading \n");
    scanf("%s", filename);

    // Open one file for reading
    fptr1 = fopen(filename, "r");
    if (fptr1 == NULL) {
        printf("Cannot open file %s \n", filename);
        exit(0);
    }

    printf("Enter the filename to open for writing \n");
    scanf("%s", filename);

    // Open another file for writing
    fptr2 = fopen(filename, "w");
    if (fptr2 == NULL) {
        printf("Cannot open file %s \n", filename);
        exit(0);
    }

    // Read contents from file
    c = fgetc(fptr1);
    while (c != EOF) {
        fputc(c, fptr2);
        c = fgetc(fptr1);
    }

    printf("\nContents copied to %s", filename);

    fclose(fptr1);
    fclose(fptr2);
    return 0;
}
```

Figure 13 Code for manual approach.

```c
//Copy the content of one text file to another text file
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp1, *fp2;
    char ch;
    fp1 = fopen("file1.txt", "r");
    if (fp1 == NULL)
    {
        printf("Cannot open file \n");
        exit(0);
    }
    fp2 = fopen("file2.txt", "w");
    if (fp2 == NULL)
    {
        printf("Cannot open file \n");
        exit(0);
    }
    ch = fgetc(fp1);
    while (ch != EOF)
    {
        fputc(ch, fp2);
        ch = fgetc(fp1);
    }
    printf("Contents copied to file2.txt");
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

Figure 14 Code for Copilot approach.

5.0 Task C

I have been using GitHub Copilot for nearly a month. GitHub Copilot, in my opinion, is fantastic! GitHub Copilot took some getting used to at first, but it now makes me so productive. GitHub Copilot will not write the program entirely, but it will help speed things up. GitHub Copilot will almost always finish the sentences perfectly. In many cases, it also writes out entire functions with either a lot of trivial code or this kind of logical gymnastics where normally programmer have to flex the brain a little, and it usually does it flawlessly. It also saved me time because it already knew what I was about to do and seemed to know the property names and query parameters that I needed. I use the Copilot approach to write my other coding assignments and projects in Visual Studio Code, and it is extremely helpful even here. I am simply ecstatic and impressed with how well the GitHub Copilot works. It's truly incredible.

I believe that the GitHub Copilot tool will assist students and programmers in coding confidently in unfamiliar territory. Try something new or code in new languages, and let GitHub Copilot suggest syntax and code in dozens of languages so programmer can spend more time learning by doing. That means giving students and programmers more time and space to work on bigger problems and building better software.

6.0 Conclusion

To summarise, it appears that developers will be writing less and less code in the coming years, if not a decade. Copilot is just one of many tools that have improved and will continue to improve our work. In fact, relying too heavily on such tools may result in unnecessary work or even serious problems. It is nowhere near replacing programmers. Nonetheless, the computer is simply superior to humans when it comes to writing boilerplate code, algorithms, and performing calculations. Let computer do it and concentrate on the creative problems.