CPS1011 Assignment Documentation

By: Aiden Schembri

ID: 205505L

Table of Contents

Section 1: Intro

Question 1:

In Question 1 the task was to create 2 algorithms to give the user the option to work out both the Secant Method and the Newton-Raphson Method of an equation, to find the closest real value. Here the user was first prompted to input the curve or line, then asked to input the first root of the equation known as x0, and the number of iterations the user would like to do. These user prompts were done in this way because they are used in both Newton-Raphson and Secant Methods.

After the user is prompted with a menu to choose between Newton-Raphson and Secant methods respectively, if the user chooses Newton-Raphson and a root is found it will be displayed, if not it will give the user an error. For the Secant Method, the user is first prompted to enter the second root of the number, whereby after, if a root is found, it is worked out and gives the user the output accordingly, and if not, the user is given an error. After each of the methods is worked out the user is then taken back to the menu where they can either choose to work out any of the methods mentioned or exit the program.

Question 2:

In Question 2 the task was to create an option for the user to enter multiple sets and allow the user to do multiple functions, such as:

- Initialise a set, allocate memory to a set.
- Deinitialise a set, deallocate the memory given prior to the set.
- Add elements to the set.
- Unionise two sets, meaning to create a set containing all unique elements from both sets.
- Intersect two sets, meaning to create a set containing all common elements shared by both sets.
- Display all the sets.
- Create a new set containing the difference between two sets.
- Count the number of elements in a set.
- Tells the user whether the first set is a subset of the second set.
- Checks whether a set is empty.

In the second part of Question 2 the user is given the option to export the set to a text file, which contains no order.

A shared library is also provided for both part A and part B of this question to allow for dynamically loadable modules, dynamic linking and code reusability.

Section 2: The Solution

Question 1:

Provided below is all the code needed for the first task including explanation of code snippets.

Question1.c:

```c
#include <stdio.h>
#include <math.h> // Used for powl and isnan

void menu(int *a, int *b, int *c, int *d, int *e, int *f, long double *x0,
int iterations); // Method declaration
void secantMethod(int *a, int *b, int *c, int *d, int *e, int *f, long
double *x0, int iterations); // Method declaration
void newtonRaphsonMethod(int *a, int *b, int *c, int *d, int *e, int *f,
long double *x0, int iterations); // Method declaration
long double differentiation(const int *a, const int *b, const int *c, const
int *d, const int *e, const long double *x0); // Method declaration
long double functionWorkOut(const int *a, const int *b, const int *c, const
int *d, const int *e, const int *f, const long double *x0); // Method
declaration

/**
 * The following main method is used to run the program.
 * It asks the user for the inputs which can be used in both Secant and
Newton-Raphson methods.
 * Returns 0 if the program is successful.
 */
int main() {
    int a, b, c, d, e, f;
    int iterations;
    long double x0;

    // Clears buffer in cases for when terminal does not output anything
    setbuf(stdout, 0);

    printf("Enter the polynomial in the format:
Ax^5+Bx^4+Cx^3+Dx^2+Ex+F=0\n");

    printf("\n");

    printf("Enter a: ");

    // While loop to check if the user has entered the correct format
    while (scanf("%d", &a) != 1){
        printf("\n");
        printf("Please enter an integer.\n");
        printf("Enter a: ");
        while (getchar() != '\n');
    }

    printf("\n");

    printf("Enter b: ");
    // While loop to check if the user has entered the correct format
    while (scanf("%d", &b) != 1){
        printf("\n");
        printf("Please enter an integer.\n");
        printf("Enter b: ");
        while (getchar() != '\n');
    }

    printf("\n");

    printf("Enter c: ");

    // While loop to check if the user has entered the correct format
    while (scanf("%d", &c) != 1){
        printf("\n");
        printf("Please enter an integer.\n");
        printf("Enter c: ");
        while (getchar() != '\n');
    }

    printf("\n");

    printf("Enter d: ");

    // While loop to check if the user has entered the correct format
    while (scanf("%d", &d) != 1){
        printf("\n");
        printf("Please enter an integer.\n");
        printf("Enter d: ");
        while (getchar() != '\n');
    }

    printf("\n");

    printf("Enter e: ");

    // While loop to check if the user has entered the correct format
    while (scanf("%d", &e) != 1){
        printf("\n");
        printf("Please enter an integer.\n");
        printf("Enter e: ");
        while (getchar() != '\n');
    }

    printf("\n");

    printf("Enter f: ");

    // While loop to check if the user has entered the correct format
    while (scanf("%d", &f) != 1){
        printf("\n");
        printf("Please enter an integer.\n");
        printf("Enter f: ");
        while (getchar() != '\n');
    }

    printf("\n");

    printf("The polynomial is: f(x)=%dx^5+%dx^4+%dx^3+%dx^2+%dx+%d\n", a,
b, c, d, e, f);
```

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

    printf("Enter the value of x0: ");

    // While loop to check if the user has entered the correct format
    while (scanf("%Lf", &x0) != 1){
        printf("\n");
        printf("Please enter any double value.\n");
        printf("Enter the value of x0: ");
        while (getchar() != '\n');
    }

    printf("\n");

    printf("Enter number of iterations: ");

    // While loop to check if the user has entered the correct format
    while (scanf("%d", &iterations) != 1){
        printf("\n");
        printf("Please enter an integer.\n");
        printf("Enter number of iterations: ");
        while (getchar() != '\n');
    }

    printf("\n");

    menu(&a, &b, &c, &d, &e, &f, &x0, iterations); // Calls menu and passes
the values of a, b, c, d, e, f, x0 and iterations

    return 0;
}

/**
 * The following method is used to display the menu.
 * Does not return anything.
 */
void menu(int *a, int *b, int *c, int *d, int *e, int *f, long double *x0,
int iterations){
    int choice;

    printf("1. Secant Method\n");
    printf("2. Newton-Raphson Method\n");
    printf("3. Quit\n");
    printf("Enter your choice: ");

    // While loop to check if the user has entered the correct format
    while (scanf("%d", &choice) != 1){
        printf("\n");
        printf("Please enter a different choice.\n");
        printf("Enter your choice: ");
        while (getchar() != '\n');
    }

    printf("\n");

    // Switch statement to determine which method to use depending on the
user's choice
    switch (choice){
        case 1:
            secantMethod(a, b, c, d, e, f, x0, iterations);
            break;
        case 2:
            newtonRaphsonMethod(a, b, c, d, e, f, x0, iterations);
            break;
        case 3:
            printf("Thank you for using this program. Goodbye!\n");
            break;
        default:
            printf("Invalid choice from menu.\n");
            menu(a, b, c, d, e, f, x0, iterations);
    }
}

/**
 * The following method is used to work out the Secant method of the
polynomial.
 * Does not return anything.
 */
void secantMethod(int *a, int *b, int *c, int *d, int *e, int *f, long
double *x0, int iterations){

    // If the iterations is less than or equal to 0 then the following if
condition will be executed
    if (iterations <= 0) {
        printf("Cannot converge with 0 or negative iterations.\n");
        printf("\n");
        menu(a, b, c, d, e, f, x0, iterations);
        return;
    }

    long double x1;

    printf("Secant Method\n");

    printf("\n");

    printf("Enter the value of x1: ");

    // While loop to check if the user has entered the correct format
    while (scanf("%Lf", &x1) != 1){
        printf("\n");
        printf("Please enter any double value.\n");
        printf("Enter a: ");
        while (getchar() != '\n');
    }

    printf("\n");

    long double doubleReturnX0 = *x0; // Used to store the original value
of x0

    long double tempX0 = *x0;
    int tempIterations = iterations; // Used to store the original value of
iterations

    do {
        long double functionValueX0 = functionWorkOut(a, b, c, d, e, f,
x0); // Work out the function with the value at x0
        long double functionValueX1 = functionWorkOut(a, b, c, d, e, f,
&x1); // Work out the function with the value at x1
```

```
        *x0 = x1 - (functionValueX1 * (x1 - *x0)) / (functionValueX1 -
functionValueX0); // Works out new x0 by getting x1 - (function at x1 * (x1
- x0)) / (function at x1 - function at x0)

        x1 = tempX0; // Set x1 to the value of x0
        tempX0 = *x0; // Set tempX0 to the value of x0

        iterations--; // Decrease iterations by 1

        // If the root is not defined then the following if condition will
be executed
        if (isnan(*x0)){
            printf("No root has been found.\n");
            printf("\n");
            menu(a, b, c, d, e, f, x0, iterations);
            return;
        }

    } while (iterations > 0);

    printf("The closest root after %d iterations is: %Lf\n",
tempIterations, *x0);
    printf("\n");

    menu(a, b, c, d, e, f, &doubleReturnX0, tempIterations); // Calls menu
and passes the original value of x0 and iterations
}

/**
 * The following method is used to work out the Newton-Raphson method of
the polynomial.
 * Does not return anything.
 */
void newtonRaphsonMethod(int *a, int *b, int *c, int *d, int *e, int *f,
long double *x0, int iterations){

    // If the iterations is less than or equal to 0 then the following if
condition will be executed
    if (iterations <= 0) {
        printf("Cannot converge with 0 or negative iterations.\n");
        printf("\n");
        menu(a, b, c, d, e, f, x0, iterations);
        return;
    }

    printf("Newton-Raphson Method\n");

    long double tempX0 = *x0; // Used to store the original value of x0
    int tempIterations = iterations; // Used to store the original value of
iterations

    do {
        long double functionValue = functionWorkOut(a, b, c, d, e, f, x0);
// Work out the function with the value at x0
        long double derivative = differentiation(a, b, c, d, e, x0); //
Work out the derivative with the value at x0

        *x0 -= functionValue / derivative; // Works out new x0 by getting
x0 - function at x0 / derivative at x0

        iterations--; // Decrease iterations by 1
        // If the root is not defined then the following if condition will
be executed
        if (isnan(*x0)){
            printf("No root has been found.\n");
            printf("\n");
            menu(a, b, c, d, e, f, x0, iterations);
            return;
        }
    } while (iterations > 0);

    printf("The closest root after %d iterations is: %Lf\n",
tempIterations, *x0);
    printf("\n");

    menu(a, b, c, d, e, f, &tempX0, tempIterations); // Calls menu and
passes the original value of x0 and iterations
}

/**
 * The following method is used to work out the differentiation of the
polynomial.
 * Returns a long double value.
 */
long double differentiation(const int *a, const int *b, const int *c, const
int *d, const int *e, const long double *x0){

    long double answer = 0;

    answer += *a * 5 * powl(*x0, 4);
    answer += *b * 4 * powl(*x0, 3);
    answer += *c * 3 * powl(*x0, 2);
    answer += *d * 2 * (*x0);
    answer += *e;

    return answer; // Returns the worked out polynomial
}

/**
 * The following method is used to work out the work out the function at a
given point.
 * Returns a long double value.
 */
long double functionWorkOut(const int *a, const int *b, const int *c, const
int *d, const int *e, const int *f, const long double *x0){

    long double answer = 0;

    answer += *a * powl(*x0, 5);
    answer += *b * powl(*x0, 4);
    answer += *c * powl(*x0, 3);
    answer += *d * powl(*x0, 2);
    answer += *e * (*x0);
    answer += *f;

    return answer; // Returns the worked out polynomial
}
```

Below one can find an explanation of the above code snippet '*Question1.c'.*

In *'Figure 1.01'*, one can notice the '#include' tags which include the needed libraries for this program to work. In this snippet we also notice all the defined functions that will be used in the program (other than the main method):

- Menu
- secantMethod
- newtonRaphsonMethod
- differentiation
- functionWorkOut

```
- Figure 1.01
#include <stdio.h>
#include <math.h> // Used for powl and isnan

void menu(int *a, int *b, int *c, int *d, int *e, int *f, long double *x0,
int iterations); // Method declaration

void secantMethod(int *a, int *b, int *c, int *d, int *e, int *f, long
double *x0, int iterations); // Method declaration

void newtonRaphsonMethod(int *a, int *b, int *c, int *d, int *e, int *f,
long double *x0, int iterations); // Method declaration

long double differentiation(const int *a, const int *b, const int *c, const
int *d, const int *e, const long double *x0); // Method declaration

long double functionWorkOut(const int *a, const int *b, const int *c, const
int *d, const int *e, const int *f, const long double *x0); // Method
declaration
```

In the code snippet titled *'Figure 1.02'*, one can find the start of the main method which includes the initialisation of the variables that will be used in the code and code needed to clear the terminal at the beginning of the run, this is done to ensure that the rest of the application runs smoothly and displays everything as intended.

```
- Figure 1.02
int main() {
    int a, b, c, d, e, f;
    int iterations;
    long double x0;

    // Clears buffer in cases for when terminal does not output anything
    setbuf(stdout, 0);
```

In *'Figure 1.03'*, we ask the user to enter a polynomial up to the 5th power by giving the user a certain format to input and then asking him to input the values they want in the places A-F.

At each step we check if the user has entered the correct format for the value they want to replace. If the user is found to have entered the correct format the program will continue, else it will keep on repeating to enter the correct format and clearing the console until the user completes this.

```
- Figure 1.03

printf("Enter the polynomial in the format: Ax^5+Bx^4+Cx^3+Dx^2+Ex+F=0\n");

printf("\n");

printf("Enter a: ");

// While loop to check if the user has entered the correct format
while (scanf("%d", &a) != 1){
    printf("\n");
    printf("Please enter an integer.\n");
    printf("Enter a: ");
    while (getchar() != '\n');
}

printf("\n");

printf("Enter b: ");

// While loop to check if the user has entered the correct format
while (scanf("%d", &b) != 1){
    printf("\n");
    printf("Please enter an integer.\n");
    printf("Enter b: ");
    while (getchar() != '\n');
}

printf("\n");

printf("Enter c: ");

// While loop to check if the user has entered the correct format
while (scanf("%d", &c) != 1){
    printf("\n");
    printf("Please enter an integer.\n");
    printf("Enter c: ");
    while (getchar() != '\n');
}

printf("\n");

printf("Enter d: ");

// While loop to check if the user has entered the correct format
while (scanf("%d", &d) != 1){
    printf("\n");
    printf("Please enter an integer.\n");
    printf("Enter d: ");
    while (getchar() != '\n');
}

printf("\n");

printf("Enter e: ");
```

```c
// While loop to check if the user has entered the correct format
while (scanf("%d", &e) != 1){
    printf("\n");
    printf("Please enter an integer.\n");
    printf("Enter e: ");
    while (getchar() != '\n');
}

printf("\n");

printf("Enter f: ");

// While loop to check if the user has entered the correct format
while (scanf("%d", &f) != 1){
    printf("\n");
    printf("Please enter an integer.\n");
    printf("Enter f: ");
    while (getchar() != '\n');
}

printf("\n");

printf("The polynomial is: f(x)=%dx^5+%dx^4+%dx^3+%dx^2+%dx+%d\n", a, b, c,
d, e, f);
```

In *'Figure 1.04'*, the user is asked to enter the first initial estimate of the root, x_0, which can be used in both the Secant Method and the Newton-Raphson Method. Present here is also the error checking to see if the user has entered the correct format in the input.

```c
- Figure 1.04
printf("Enter the value of x0: ");

// While loop to check if the user has entered the correct format
while (scanf("%Lf", &x0) != 1){
    printf("\n");
    printf("Please enter any double value.\n");
    printf("Enter the value of x0: ");
    while (getchar() != '\n');
}
```

In *'Figure 1.05'*, the user is asked to enter the number of iterations that they would like to use for any of the 2 methods to get the accuracy they need. Here we can also see error checking to make sure that the user is unable to enter the wrong format in the input.

After the number of iterations are entered, the user is then taken to the menu page by calling the menu function, and in the parameters the pass by reference is used to pass the address of the variables instead of copying the values inside of the variables.

```c
- Figure 1.05
printf("Enter number of iterations: ");

// While loop to check if the user has entered the correct format
```

```
while (scanf("%d", &iterations) != 1){
    printf("\n");
    printf("Please enter an integer.\n");
    printf("Enter number of iterations: ");
    while (getchar() != '\n');
}

printf("\n");

menu(&a, &b, &c, &d, &e, &f, &x0, iterations); // Calls menu and passes the values of a, b, c, d, e, f, x0 and iterations

return 0;
```

In *'Figure 1.06'*, we can see the whole menu function. Here the user is given 3 options to choose from:

1. Secant Method
2. Newton-Raphson Method
3. Exit

If the user enters the number '1', they are taken to the function which works the Secant Method called 'secantMethod', in the parameters here the pass by reference is also used. If the user enters the number '2' they are taken to the function that works out the Newton-Raphson Method called 'newtonRaphsonMethod', pass by reference is used in the parameters. If the user enters the number '3' the system outputs a Goodbye prompt, and the user is taken back to the main function where the program will exit and return 0. If the user does not enter any of the above inputs' recursion is used by calling the menu again, this eliminates the need for any loops like do... while loops, and again pass by reference is used here. As one can notice here during the input, we also have error checking to ensure that the user is only able to enter integers and no other format; if the user does not enter the correct format, they are continuously prompted to enter the correct format until they do so.

```
- Figure 1.06
void menu(int *a, int *b, int *c, int *d, int *e, int *f, long double *x0, int iterations){
    int choice;

    printf("1. Secant Method\n");
    printf("2. Newton-Raphson Method\n");
    printf("3. Quit\n");
    printf("Enter your choice: ");

    // While loop to check if the user has entered the correct format
    while (scanf("%d", &choice) != 1){
        printf("\n");
        printf("Please enter a different choice.\n");
        printf("Enter your choice: ");
        while (getchar() != '\n');
    }

    printf("\n");
```

```c
    // Switch statement to determine which method to use depending on the user's choice
    switch (choice){
        case 1:
            secantMethod(a, b, c, d, e, f, x0, iterations);
            break;
        case 2:
            newtonRaphsonMethod(a, b, c, d, e, f, x0, iterations);
            break;
        case 3:
            printf("Thank you for using this program. Goodbye!\n");
            break;
        default:
            printf("Invalid choice from menu.\n");
            menu(a, b, c, d, e, f, x0, iterations);
    }
}
```

In *'Figure 1.07'*, one can witness the method used to work the Secant Method called, secantMethod. In it we first do error handling to check if the user has entered any number of iterations greater than 0 as without this the program would not work and crash, so to avoid this we check and output an error to the user and we take the user back to the menu.

After we ask the user to enter x_1 which is the second approximation to the root function. Here error handling is also used to ensure that the user has entered the correct format of x_1, to make sure that the program does not crash.

After the second approximation is entered the program proceeds to work out the closest approximation using the Secant Method:

$$x_{n+1} = x_n - \frac{f(x_n).(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

The algorithm within the code first proceeds to work out x_{n-1} by calling the function workout method and places that in a variable and does the same with $x_{n,}$ and after places the secant method worked out in a pointer and removes 1 from the iterations. If at any point in the do… while loop the root of the answer of the worked-out secant method becomes NaN (Not a number), this will output an error as this means that there is no root for the values that the user has entered and will take the user back to the menu.

If the secant method is worked out correctly it will be outputted for the user to the nearest 6 decimal places. The user is then taken back to the menu.

```
- Figure 1.07
void secantMethod(int *a, int *b, int *c, int *d, int *e, int *f, long
double *x0, int iterations){

    // If the iterations is less than or equal to 0 then the following if
condition will be executed
    if (iterations <= 0) {
        printf("Cannot converge with 0 or negative iterations.\n");
        printf("\n");
        menu(a, b, c, d, e, f, x0, iterations);
        return;
    }

    long double x1;

    printf("Secant Method\n");

    printf("\n");

    printf("Enter the value of x1: ");

    // While loop to check if the user has entered the correct format
    while (scanf("%Lf", &x1) != 1){
        printf("\n");
        printf("Please enter any double value.\n");
        printf("Enter a: ");
        while (getchar() != '\n');
    }

    printf("\n");

    long double doubleReturnX0 = *x0; // Used to store the original value
of x0

    long double tempX0 = *x0;
    int tempIterations = iterations; // Used to store the original value of
iterations

    do {
        long double functionValueX0 = functionWorkOut(a, b, c, d, e, f,
x0); // Work out the function with the value at x0
        long double functionValueX1 = functionWorkOut(a, b, c, d, e, f,
&x1); // Work out the function with the value at x1

        *x0 = x1 - (functionValueX1 * (x1 - *x0)) / (functionValueX1 -
functionValueX0); // Works out new x0 by getting x1 - (function at x1 * (x1
- x0)) / (function at x1 - function at x0)

        x1 = tempX0; // Set x1 to the value of x0
        tempX0 = *x0; // Set tempX0 to the value of x0

        iterations--; // Decrease iterations by 1

        // If the root is not defined then the following if condition will
be executed
        if (isnan(*x0)){
            printf("No root has been found.\n");
            printf("\n");
            menu(a, b, c, d, e, f, x0, iterations);
            return;
        }

    } while (iterations > 0);

    printf("The closest root after %d iterations is: %Lf\n",
tempIterations, *x0);
    printf("\n");

    menu(a, b, c, d, e, f, &doubleReturnX0, tempIterations); // Calls menu
and passes the original value of x0 and iterations
}
```

In *'Figure 1.08'*, one can see the function that works out the Newton-Raphson method, called newtonRaphsonMethod. Here the algorithm, as done in the secant function, first checks if the number of iterations is greater than 0 and outputs an error if the value is not greater than 0.

If the value of the iterations is found to be suitable the algorithm to work out the Newton-Raphson Method starts. First the function is worked out with the current value of x_n by calling the functionWorkOut function and it is placed in a variable. After the differentiation of the polynomial is worked out together with the value of x_n and the value is returned into a variable.

After, the algorithm works out the Newton-Raphson value at the current value of x by using the equation:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

After working this out the number of iterations is removed by 1 and the algorithm checks if the worked-out number is classified as NaN (Not a Number) and if so it will output an error saying that no root has been found and the user is taken back to the menu.

If a closest root is found this will be outputted to the user at the end of the function up to 6 decimal places and the user is taken back to the menu.

```
- Figure 1.08
void newtonRaphsonMethod(int *a, int *b, int *c, int *d, int *e, int *f,
long double *x0, int iterations){

    // If the iterations is less than or equal to 0 then the following if
condition will be executed
    if (iterations <= 0) {
        printf("Cannot converge with 0 or negative iterations.\n");
        printf("\n");
        menu(a, b, c, d, e, f, x0, iterations);
        return;
    }

    printf("Newton-Raphson Method\n");
    long double tempX0 = *x0; // Used to store the original value of x0
    int tempIterations = iterations; // Used to store the original value of
iterations

    do {
        long double functionValue = functionWorkOut(a, b, c, d, e, f, x0);
// Work out the function with the value at x0
        long double derivative = differentiation(a, b, c, d, e, x0); //
Work out the derivative with the value at x0

        *x0 -= functionValue / derivative; // Works out new x0 by getting
x0 - function at x0 / derivative at x0

        iterations--; // Decrease iterations by 1

        // If the root is not defined then the following if condition will
be executed
        if (isnan(*x0)){
            printf("No root has been found.\n");
            printf("\n");
            menu(a, b, c, d, e, f, x0, iterations);
            return;
        }
    } while (iterations > 0);

    printf("The closest root after %d iterations is: %Lf\n",
tempIterations, *x0);
    printf("\n");

    menu(a, b, c, d, e, f, &tempX0, tempIterations); // Calls menu and
passes the original value of x0 and iterations
}
```

In *'Figure 1.09'*, we can see the function that works out the differentiation of the function. Here the program gets the values inside of the respective addresses and work out the function at the current value of x. It then returns the answer of the function.

```
- Figure 1.09
long double differentiation(const int *a, const int *b, const int *c, const
int *d, const int *e, const long double *x0){

    long double answer = 0;

    answer += *a * 5 * powl(*x0, 4);
    answer += *b * 4 * powl(*x0, 3);
    answer += *c * 3 * powl(*x0, 2);
    answer += *d * 2 * (*x0);
    answer += *e;

    return answer; // Returns the worked out polynomial
}
```

In *'Figure 1.10'*, we see almost the work out function where the function at the current value of x is worked out and the answer is returned for the user.

```
- Figure 1.10
long double functionWorkOut(const int *a, const int *b, const int *c, const
int *d, const int *e, const int *f, const long double *x0){

    long double answer = 0;

    answer += *a * powl(*x0, 5);
    answer += *b * powl(*x0, 4);
    answer += *c * powl(*x0, 3);
    answer += *d * powl(*x0, 2);
    answer += *e * (*x0);
    answer += *f;

    return answer; // Returns the worked out polynomial
}
```

<u>CMakeLists.txt</u>

```
cmake_minimum_required(VERSION 3.26)
project("Assignment Question 1" C)

set(CMAKE_C_STANDARD 11)

MATH(EXPR stack_size "16 * 1024 * 1024") # 16 Mb
set(CMAKE_EXE_LINKER_FLAGS "-Wl,--stack,${stack_size}")

add_executable(Question_1 Question1.c)
```

Below will be an explanation of what parts of the above provided code do.

In *'Figure 2.1'*, we are defining the name to be given to the project. Here the name is put in quotation marks to allow whitespaces to be used in the project name.

```
- Figure 2.1
project("Assignment Question 1" C)
```

The 2 lines shown in *'Figure 2.2'* are only used on operating systems like Windows, which require the stack size to be increased for the algorithm in the code to work, as the stack size predefined on Windows is too small and will crash the program. On other operating systems like Mac OS, this code snippet will need to be commented out by placing a '#' in front of the two lines as follows in *'Figure 2.3'*:

```
- Figure 2.2
MATH(EXPR stack_size "16 * 1024 * 1024") # 16 Mb
set(CMAKE_EXE_LINKER_FLAGS "-Wl,--stack,${stack_size}")
```

```
- Figure 2.3
#MATH(EXPR stack_size "16 * 1024 * 1024") # 16 Mb
#set(CMAKE_EXE_LINKER_FLAGS "-Wl,--stack,${stack_size}")
```

In *'Figure 2.4'*, we will now find the part of the code that will create a code snippet for us to be able to execute the program in this task.

```
- Figure 2.4
add_executable(Question_1 Question1.c)
```

Question 2A:

Provided below is all the code needed for the task 2A including explanation of code snippets.

Question2A.c

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

void userDefinedSets(); // User defined sets
void predefinedSets(); // Predefined sets

int main() {
    int mainChoice;

    printf("Welcome to the Set Operations Program!\n");

    // Menu
    do {
        printf("\n1.Enter your own sets");
        printf("\n2.Use the predefined sets");
        printf("\n3.Exit");
        printf("\n\nEnter your choice: ");

        while (scanf("%d", &mainChoice) != 1) {
            printf("Invalid input. Please enter an integer.\n");
            printf("\n\nEnter your choice: ");
            while (getchar() != '\n'); // Clear input buffer
        }

        while (getchar() != '\n'); // Clear input buffer
        // Switch case for main menu
        switch (mainChoice) {
            case 1:
                printf("\n");
                userDefinedSets();
                break;
            case 2:
                printf("\n");
                predefinedSets();
                break;
            case 3:
                break;
            default:
                printf("\nInvalid choice. Please try again.\n");
                break;
        }
    }while(mainChoice != 3);

    printf("\nThank you for using this program. Goodbye!\n");

    return 0;
}


void userDefinedSets(){
    GenSet intSet, stringSet, stringSet2, intSet2;
    initSet(&intSet, 0); // Integer set
    initSet(&intSet2, 0); // Integer set
    initSet(&stringSet, 1); // String set
    initSet(&stringSet2, 1); // String set

    int intElement;
    int choice;
    char stringElement[MAX_STRING_LENGTH];

    printf("Adding elements to the First Integer Set:\n");
    printf("Enter integers (press Enter without typing anything to stop):
");

    // While loop to get user input
    while (1) {
        char line[MAX_SET_SIZE];
        if (!fgets(line, sizeof(line), stdin)) {
            // Error reading input
            printf("Error reading input. Exiting.\n");
            break;
        }

        // Remove the trailing newline character, if present
        line[strcspn(line, "\n")] = '\0';

        if (line[0] == '\0') {
            // Stop loop if the user presses Enter without typing anything
            break;
        }

        // Convert the input to an integer
        if (sscanf(line, "%d", &intElement) != 1) {
            printf("Invalid input. Please enter an integer.\n");
            continue; // Skip adding to the set if input is not an integer
        }

        addToSet(&intSet, &intElement);
    }


    printf("\nAdding elements to the First String Set:\n");
    printf("Enter strings (press 'Enter' to stop): ");
    // While loop to get user input
    while (1) {
        if (!fgets(stringElement, sizeof(stringElement), stdin)) {
            // Error reading input
            printf("Error reading input. Exiting.\n");
            break;
        }

        // Remove the trailing newline character, if present
        stringElement[strcspn(stringElement, "\n")] = '\0';

        if (stringElement[0] == '\0') {
            // Stop loop if the user presses Enter without typing anything
            break;
        }

        addToSet(&stringSet, stringElement);

        // Check for end-of-file
        if (feof(stdin)) {
            break;
        }
    }

    printf("Adding elements to the Second Integer Set:\n");
    printf("Enter integers (press Enter without typing anything to stop):
");

    // While loop to get user input
    while (1) {
        char line[MAX_SET_SIZE];
        if (!fgets(line, sizeof(line), stdin)) {
            // Error reading input
            printf("Error reading input. Exiting.\n");
            break;
        }

        // Remove the trailing newline character, if present
        line[strcspn(line, "\n")] = '\0';

        if (line[0] == '\0') {
            // Stop loop if the user presses Enter without typing anything
            break;
        }

        // Convert the input to an integer
        if (sscanf(line, "%d", &intElement) != 1) {
            printf("Invalid input. Please enter an integer.\n");
            continue; // Skip adding to the set if input is not an integer
        }

        addToSet(&intSet2, &intElement);
    }

    printf("\nAdding elements to the Second String Set:\n");
    printf("Enter strings (press 'Enter' to stop): ");
    // While loop to get user input
    while (1) {
        if (!fgets(stringElement, sizeof(stringElement), stdin)) {
            // Error reading input
            printf("Error reading input. Exiting.\n");
            break;
        }

        // Remove the trailing newline character, if present
        stringElement[strcspn(stringElement, "\n")] = '\0';

        if (stringElement[0] == '\0') {
            // Stop loop if the user presses Enter without typing anything
            break;
        }

        addToSet(&stringSet2, stringElement);

        // Check for end-of-file
        if (feof(stdin)) {
            break;
        }
    }

    printf("\nInteger Set:\n");
    displaySet(intSet);
    printf("\nString Set:\n");
    displaySet(stringSet);
    printf("\nInt Set 2:\n");
    displaySet(intSet2);
    printf("\nString Set 2:\n");
    displaySet(stringSet2);

    // Menu
    do{
        printf("\n\nMenu:");
        printf("\n1. Display integer set");
        printf("\n2. Display string set");
        printf("\n3. Display integer set 2");
        printf("\n4. Display string set 2");
        printf("\n5. Count integer set");
        printf("\n6. Count string set");
        printf("\n7. Check if integer set is empty");
        printf("\n8. Check if string set is empty");
        printf("\n9. Check if integer element is a subset of the integer
set");
        printf("\n10. Check if string element is a subset of the string
set");
        printf("\n11. Union of integer set and integer set 2");
        printf("\n12. Intersection of string set and string set 2");
        printf("\n13. Difference between integer set and integer set 2");
        printf("\n14. Exit");
        printf("\n\nEnter your choice: ");

        while (scanf("%d", &choice) != 1) {
            printf("Invalid input. Please enter an integer.\n");
            printf("\n\nEnter your choice: ");
            while (getchar() != '\n'); // Clear input buffer
        }

        switch(choice){
            case 1:
                printf("\nInteger Set:\n");
                displaySet(intSet); // Display integer set
                break;
            case 2:
                printf("\nString Set:\n");
                displaySet(stringSet); // Display string set
                break;
            case 3:
                printf("\nInteger Set 2:\n");
                displaySet(intSet2); // Display integer set 2
                break;
            case 4:
                printf("\nString Set 2:\n");
                displaySet(stringSet2); // Display string set 2
                break;
            case 5:
                printf("\nCount of elements in Integer Set: %d",
countSet(intSet)); // Count of elements in integer set
                break;
            case 6:
                printf("\nCount of elements in String Set: %d",
countSet(stringSet)); // Count of elements in string set
                break;
            case 7:
                printf("\nInteger set is empty: %s", isEmptySet(intSet) ?
"Yes" : "No"); // Check if integer set is empty
                break;
            case 8:
                printf("\nString set is empty: %s", isEmptySet(stringSet) ?
"Yes" : "No"); // Check if string set is empty
                break;
            case 9:
                printf("\nEnter an integer: ");

                while (scanf("%d", &intElement) != 1) {
                    printf("Invalid input. Please enter an integer.\n");
                    printf("\nEnter an integer: ");
                    while (getchar() != '\n'); // Clear input buffer
                }
                printf("\nIs '%d' a found in the integer set: %s",
intElement, isSubsetSet(intSet, &intElement) ? "Yes" : "No"); // Check if
integer set is a subset of another integer set
                break;
            case 10:
                printf("\nEnter a string: ");
                while (scanf("%s", stringElement) != 1) {
                    printf("Invalid input. Please enter a string.\n");
                    printf("\nEnter a string: ");
                    while (getchar() != '\n'); // Clear input buffer
                }
                printf("\nIs '%s' found in the string set: %s",
stringElement, isSubsetSet(stringSet, stringElement) ? "Yes" : "No"); //
Check if string set is a subset of another string set
                break;
            case 11:
                printf("\n");

                GenSet *unionResult = unionSet(intSet, intSet2); // Get
union of integer set and string set

                // Check if result is NULL
                if (unionResult != NULL) {
                    printf("\nUnion of Integer Set and Integer Set 2: ");
                    displaySet(*unionResult);
                    deinitSet(unionResult);
                } else {
                    printf("Error getting unionSet result.\n");
                }

                break;
            case 12:
                printf("\n");

                GenSet *intersectResult = intersectSet(stringSet,
stringSet2); // Get intersection of integer set and string set

                // Check if result is NULL
                if(intersectResult != NULL) {
                    printf("\nIntersection of String Set and String Set 2:
");

                    displaySet(*intersectResult);
                    deinitSet(intersectResult);
                } else {
                    printf("Error getting intersectSet result.\n");
                }

                break;
            case 13:
                printf("\n");

                GenSet *diffResult = diffSet(intSet, intSet2); // Get
difference between integer set and string set

                // Check if result is NULL
                if(diffResult != NULL) {
                    printf("\nDifference between Integer Set and Integer
Set 2: ");

                    displaySet(*diffResult);
                    deinitSet(diffResult);
                } else {
                    printf("Error getting diffSet result.\n");
                }

                break;
            case 14:
                break;
            default:
                printf("\nInvalid choice. Please try again.");
                break;
        }
    } while (choice != 14);

    deinitSet(&intSet); // Deinitialize integer set
    deinitSet(&stringSet); // Deinitialize string set
}

void predefinedSets(){
    GenSet intSet, stringSet, stringSet2, intSet2;
    initSet(&intSet, 0); // Integer set
    initSet(&stringSet, 1); // String set
    initSet(&stringSet2, 1); // String set
    initSet(&intSet2, 0); // Integer set

    // Adding elements to integer set
    addToSet(&intSet, &(int){10});
```

```c
    addToSet(&intSet, &(int){20});
    addToSet(&intSet, &(int){30});
    addToSet(&intSet, &(int){40});

    // Adding of elements to integer set 2
    addToSet(&intSet2, &(int){10});

    // Adding elements to string set
    addToSet(&stringSet, "apple");
    addToSet(&stringSet, "orange");
    addToSet(&stringSet, "banana");
    addToSet(&stringSet, "apple"); // Duplicate element

    // Adding elements to string set 2
    addToSet(&stringSet2, "apple");

    printf("\nInteger Set:\n");
    displaySet(intSet);
    printf("\nString Set:\n");
    displaySet(stringSet);

    printf("\nCount of elements in Integer Set: %d", countSet(intSet)); //
Count of elements in integer set
    printf("\nCount of elements in String Set: %d", countSet(stringSet));
// Count of elements in string set

    printf("\n\nChecking if Integer Set is empty: %s", isEmptySet(intSet) ?
"Yes" : "No"); // Check if integer set is empty
    printf("\nChecking if String Set is empty: %s", isEmptySet(stringSet) ?
"Yes" : "No"); // Check if string set is empty

    printf("\n\nIs 20 a subset of Integer Set: %s", isSubsetSet(intSet,
&(int){20}) ? "Yes" : "No"); // Check if 20 is a subset of integer set
    printf("\nIs 'apple' a subset of String Set: %s",
isSubsetSet(stringSet, "apple") ? "Yes" : "No"); // Check if 'apple' is a
subset of string set
    printf("\nIs 'kiwi' a subset of String Set: %s", isSubsetSet(stringSet,
"kiwi") ? "Yes" : "No"); // Check if 'kiwi' is a subset of string set

    printf("\n");

    GenSet *intStringUnion = unionSet(intSet, stringSet); // Get union of
integer set and string set

    // Check if result is NULL
    if(intStringUnion != NULL) {
        printf("\n\nUnion of Integer Set and String Set: ");
        displaySet(*intStringUnion);
        deinitSet(intStringUnion);
    } else {
        printf("Error getting unionSet result.\n");
    }

    printf("\n");

    GenSet *intStringIntersect = intersectSet(intSet, stringSet); // Get
intersection of integer set and string set

    // Check if result is NULL
    if(intStringIntersect != NULL) {
        printf("\nIntersection of Integer Set and String Set: ");
        displaySet(*intStringIntersect);
        deinitSet(intStringIntersect);
    } else {
        printf("Error getting intersectSet result.\n");
    }

    printf("\n");

    GenSet *stringSetIntersection = intersectSet(stringSet, stringSet2); //
Get intersection of string set and string set 2

    // Check if result is NULL
    if(stringSetIntersection != NULL) {
        printf("\nIntersection of String Set and String Set 2: ");
        displaySet(*stringSetIntersection);
        deinitSet(stringSetIntersection);
    } else {
        printf("Error getting intersectSet result.\n");
    }

    printf("\n");

    GenSet *intSetIntersection = intersectSet(intSet, intSet2); // Get
intersection of integer set and integer set 2

    // Check if result is NULL
    if(intSetIntersection != NULL) {
        printf("\nIntersection of Int Set and Int Set 2: ");
        displaySet(*intSetIntersection);
        deinitSet(intSetIntersection);
    } else {
        printf("Error getting intersectSet result.\n");
    }

    printf("\n");

    GenSet *intStringDiff = diffSet(intSet, stringSet); // Get difference
between integer set and string set

    // Check if result is NULL
    if(intStringDiff != NULL) {
        printf("\nDifference between Integer Set and String Set: ");
        displaySet(*intStringDiff);
        deinitSet(intStringDiff);
    } else {
        printf("Error getting diffSet result.\n");
    }

    deinitSet(&intSet); // Deinitialize integer set
    deinitSet(&stringSet); // Deinitialize string set
    deinitSet(&stringSet2); // Deinitialize string set 2
    deinitSet(&intSet2); // Deinitialize integer set 2
}
```

Below is an explanation using code snippets from the above code:

In *'Figure 3.01'*, we can see the '#include' tag which represents the libraries we will be needing in this c file, one of these libraries is the header file which we have defined called ‘setMethodsA.h’. We can also see the functions (other than main function) that we will be using in this .c file, those being:

- userDefinedSets
- predefinedSets

```
- Figure 3.01
#include <stdio.h>
#include "setMethodsA.h"
#include <string.h>

void userDefinedSets(); // User defined sets
void predefinedSets(); // Predefined sets
```

In *‘Figure 3.02’*, we can see the main function which is called when the program is run. This function contains a menu that will output for the user to let the user choose between system predefined sets or sets which the user can enter.

```
- Figure 3.02
int main() {
    int mainChoice;

    printf("Welcome to the Set Operations Program!\n");

    // Menu
    do {
        printf("\n1.Enter your own sets");
        printf("\n2.Use the predefined sets");
        printf("\n3.Exit");
        printf("\n\nEnter your choice: ");

        while (scanf("%d", &mainChoice) != 1) {
            printf("Invalid input. Please enter an integer.\n");
            printf("\n\nEnter your choice: ");
            while (getchar() != '\n'); // Clear input buffer
        }

        while (getchar() != '\n'); // Clear input buffer
        // Switch case for main menu
        switch (mainChoice) {
            case 1:
                printf("\n");
                userDefinedSets();
                break;
            case 2:
                printf("\n");
                predefinedSets();
                break;
            case 3:
                break;
            default:
                printf("\nInvalid choice. Please try again.\n");
                break;
        }
    }while(mainChoice != 3);
    printf("\nThank you for using this program. Goodbye!\n");

    return 0;
}
```

In *'Figure 3.03'*, we see the userDefinedSets function. Here the user is allowed to enter their own 2 integer and 2 string set, while checking the user enters the correct data types as required.

After the sets have been added the program moves on to displaying a menu for the user with the following functions:

1. Display integer set – The program displays the first integer set inputted by the user
2. Display string set – The program displays the first string set inputted by the user
3. Display integer set 2 – The program displays the second integer set inputted by the user
4. Display string set 2 – The program displays the second string set inputted by the user
5. Count integer set – Checks the number of elements in the first integer set
6. Count string set – Checks the number of elements in the first string set
7. Check if integer set is empty – Checks if the first integer set is empty
8. Check if string set is empty – Check if the string set is empty
9. Check if integer element is a subset of the integer set – Checks if any integer inputted by the user is part of the first integer set
10. Check if string element is a subset of the string set – Checks if any string inputted by the user is part of the string set
11. Union of integer set and integer set 2 – Does the union of first and second integer sets
12. Intersection of string set and string set 2 – Does the intersection of first and second string sets
13. Difference between integer set and integer set 2 – Checks what the difference is between the first and second integer sets
14. Exit – Exits the function.

```
- Figure 3.03
void userDefinedSets(){
    GenSet intSet, stringSet, stringSet2, intSet2;
    initSet(&intSet, 0); // Integer set
    initSet(&intSet2, 0); // Integer set
    initSet(&stringSet, 1); // String set
    initSet(&stringSet2, 1); // String set

    int intElement;
    int choice;
    char stringElement[MAX_STRING_LENGTH];

    printf("Adding elements to the First Integer Set:\n");
    printf("Enter integers (press Enter without typing anything to stop):
");

    // While loop to get user input
    while (1) {
        char line[MAX_SET_SIZE];
        if (!fgets(line, sizeof(line), stdin)) {
            // Error reading input
            printf("Error reading input. Exiting.\n");
            break;
        }

        // Remove the trailing newline character, if present
        line[strcspn(line, "\n")] = '\0';

        if (line[0] == '\0') {
            // Stop loop if the user presses Enter without typing anything
            break;
        }

        // Convert the input to an integer
        if (sscanf(line, "%d", &intElement) != 1) {
            printf("Invalid input. Please enter an integer.\n");
            continue; // Skip adding to the set if input is not an integer
        }

        addToSet(&intSet, &intElement);
    }


    printf("\nAdding elements to the First String Set:\n");
    printf("Enter strings (press 'Enter' to stop): ");
    // While loop to get user input
    while (1) {
        if (!fgets(stringElement, sizeof(stringElement), stdin)) {
            // Error reading input
            printf("Error reading input. Exiting.\n");
            break;
        }

        // Remove the trailing newline character, if present
        stringElement[strcspn(stringElement, "\n")] = '\0';

        if (stringElement[0] == '\0') {
            // Stop loop if the user presses Enter without typing anything
            break;
        }

        addToSet(&stringSet, stringElement);

        // Check for end-of-file
        if (feof(stdin)) {
            break;
        }
    }

    printf("Adding elements to the Second Integer Set:\n");
    printf("Enter integers (press Enter without typing anything to stop):
");

    // While loop to get user input
    while (1) {
        char line[MAX_SET_SIZE];
        if (!fgets(line, sizeof(line), stdin)) {
            // Error reading input
```

```c
            printf("Error reading input. Exiting.\n");
            break;
        }

        // Remove the trailing newline character, if present
        line[strcspn(line, "\n")] = '\0';

        if (line[0] == '\0') {
            // Stop loop if the user presses Enter without typing anything
            break;
        }

        // Convert the input to an integer
        if (sscanf(line, "%d", &intElement) != 1) {
            printf("Invalid input. Please enter an integer.\n");
            continue; // Skip adding to the set if input is not an integer
        }

        addToSet(&intSet2, &intElement);
    }

    printf("\nAdding elements to the Second String Set:\n");
    printf("Enter strings (press 'Enter' to stop): ");
    // While loop to get user input
    while (1) {
        if (!fgets(stringElement, sizeof(stringElement), stdin)) {
            // Error reading input
            printf("Error reading input. Exiting.\n");
            break;
        }

        // Remove the trailing newline character, if present
        stringElement[strcspn(stringElement, "\n")] = '\0';

        if (stringElement[0] == '\0') {
            // Stop loop if the user presses Enter without typing anything
            break;
        }

        addToSet(&stringSet2, stringElement);

        // Check for end-of-file
        if (feof(stdin)) {
            break;
        }
    }

    printf("\nInteger Set:\n");
    displaySet(intSet);
    printf("\nString Set:\n");
    displaySet(stringSet);
    printf("\nInt Set 2:\n");
    displaySet(intSet2);
    printf("\nString Set 2:\n");
    displaySet(stringSet2);

    // Menu
    do{
        printf("\n\nMenu:");
        printf("\n1. Display integer set");
        printf("\n2. Display string set");
        printf("\n3. Display integer set 2");
        printf("\n4. Display string set 2");
        printf("\n5. Count integer set");
        printf("\n6. Count string set");
        printf("\n7. Check if integer set is empty");
        printf("\n8. Check if string set is empty");
        printf("\n9. Check if integer element is a subset of the integer
set");
        printf("\n10. Check if string element is a subset of the string
set");
        printf("\n11. Union of integer set and integer set 2");
        printf("\n12. Intersection of string set and string set 2");
        printf("\n13. Difference between integer set and integer set 2");
        printf("\n14. Exit");
        printf("\n\nEnter your choice: ");

        while (scanf("%d", &choice) != 1) {
            printf("Invalid input. Please enter an integer.\n");
            printf("\n\nEnter your choice: ");
            while (getchar() != '\n'); // Clear input buffer
        }

        switch(choice){
            case 1:
                printf("\nInteger Set:\n");
                displaySet(intSet); // Display integer set
                break;
            case 2:
                printf("\nString Set:\n");
                displaySet(stringSet); // Display string set
                break;
            case 3:
                printf("\nInteger Set 2:\n");
                displaySet(intSet2); // Display integer set 2
                break;
            case 4:
                printf("\nString Set 2:\n");
                displaySet(stringSet2); // Display string set 2
                break;
            case 5:
                printf("\nCount of elements in Integer Set: %d",
countSet(intSet)); // Count of elements in integer set
                break;
            case 6:
                printf("\nCount of elements in String Set: %d",
countSet(stringSet)); // Count of elements in string set
                break;
            case 7:
                printf("\nInteger set is empty: %s", isEmptySet(intSet) ?
"Yes" : "No"); // Check if integer set is empty
                break;
            case 8:
                printf("\nString set is empty: %s", isEmptySet(stringSet) ?
"Yes" : "No"); // Check if string set is empty
                break;
            case 9:
                printf("\nEnter an integer: ");

                while (scanf("%d", &intElement) != 1) {
                    printf("Invalid input. Please enter an integer.\n");
                    printf("\nEnter an integer: ");
                    while (getchar() != '\n'); // Clear input buffer
                }
                printf("\nIs '%d' a found in the integer set: %s",
intElement, isSubsetSet(intSet, &intElement) ? "Yes" : "No"); // Check if
integer set is a subset of another integer set
                break;
            case 10:
                printf("\nEnter a string: ");
                while (scanf("%s", stringElement) != 1) {
                    printf("Invalid input. Please enter a string.\n");
                    printf("\nEnter a string: ");
                    while (getchar() != '\n'); // Clear input buffer
                }
                printf("\nIs '%s' found in the string set: %s",
stringElement, isSubsetSet(stringSet, stringElement) ? "Yes" : "No"); //
Check if string set is a subset of another string set
                break;
            case 11:
                printf("\n");

                GenSet *unionResult = unionSet(intSet, intSet2); // Get
union of integer set and string set

                // Check if result is NULL
                if (unionResult != NULL) {
                    printf("\nUnion of Integer Set and Integer Set 2: ");
                    displaySet(*unionResult);
                    deinitSet(unionResult);
                } else {
                    printf("Error getting unionSet result.\n");
                }

                break;
            case 12:
                printf("\n");

                GenSet *intersectResult = intersectSet(stringSet,
stringSet2); // Get intersection of integer set and string set

                // Check if result is NULL
                if(intersectResult != NULL) {
                    printf("\nIntersection of String Set and String Set 2:
");

                    displaySet(*intersectResult);
                    deinitSet(intersectResult);
                } else {
                    printf("Error getting intersectSet result.\n");
                }

                break;
            case 13:
                printf("\n");

                GenSet *diffResult = diffSet(intSet, intSet2); // Get
difference between integer set and string set

                // Check if result is NULL
                if(diffResult != NULL) {
                    printf("\nDifference between Integer Set and Integer
Set 2: ");
                    displaySet(*diffResult);
                    deinitSet(diffResult);
                } else {
                    printf("Error getting diffSet result.\n");
                }

                break;
            case 14:
                break;
            default:
                printf("\nInvalid choice. Please try again.");
                break;
        }
    } while (choice != 14);

    deinitSet(&intSet); // Deinitialize integer set
    deinitSet(&stringSet); // Deinitialize string set
}
```

In *‘Figure 3.04’*, we have the function called predefinedSets, and here, system defined sets are used to make the process faster for the user. Here some elements are added into 4 different sets:

- intSet
- stringSet
- stringSet2
- intSet2

After some general tasks are run to ensure that the functions defined in setMethodsA.c are running correctly. At the end the sets are deinitialised to make sure no memory leakages occur.

```
- Figure 3.04
void predefinedSets(){
    GenSet intSet, stringSet, stringSet2, intSet2;
    initSet(&intSet, 0); // Integer set
    initSet(&stringSet, 1); // String set
    initSet(&stringSet2, 1); // String set
    initSet(&intSet2, 0); // Integer set

    // Adding elements to integer set
    addToSet(&intSet, &(int){10});
    addToSet(&intSet, &(int){20});
    addToSet(&intSet, &(int){30});
    addToSet(&intSet, &(int){40});

    // Adding of elements to integer set 2
    addToSet(&intSet2, &(int){10});

    // Adding elements to string set
    addToSet(&stringSet, "apple");
    addToSet(&stringSet, "orange");
    addToSet(&stringSet, "banana");
    addToSet(&stringSet, "apple"); // Duplicate element

    // Adding elements to string set 2
    addToSet(&stringSet2, "apple");

    printf("\nInteger Set:\n");
    displaySet(intSet);
    printf("\nString Set:\n");
    displaySet(stringSet);

    printf("\nCount of elements in Integer Set: %d", countSet(intSet)); //
Count of elements in integer set
    printf("\nCount of elements in String Set: %d", countSet(stringSet));
// Count of elements in string set

    printf("\n\nChecking if Integer Set is empty: %s", isEmptySet(intSet) ?
"Yes" : "No"); // Check if integer set is empty
    printf("\nChecking if String Set is empty: %s", isEmptySet(stringSet) ?
"Yes" : "No"); // Check if string set is empty

    printf("\n\nIs 20 a subset of Integer Set: %s", isSubsetSet(intSet,
&(int){20}) ? "Yes" : "No"); // Check if 20 is a subset of integer set
    printf("\nIs 'apple' a subset of String Set: %s",
isSubsetSet(stringSet, "apple") ? "Yes" : "No"); // Check if 'apple' is a
subset of string set
    printf("\nIs 'kiwi' a subset of String Set: %s", isSubsetSet(stringSet,
"kiwi") ? "Yes" : "No"); // Check if 'kiwi' is a subset of string set

    printf("\n");

    GenSet *intStringUnion = unionSet(intSet, stringSet); // Get union of
integer set and string set

    // Check if result is NULL
    if(intStringUnion != NULL) {
        printf("\n\nUnion of Integer Set and String Set: ");
        displaySet(*intStringUnion);
        deinitSet(intStringUnion);
    } else {
        printf("Error getting unionSet result.\n");
    }

    printf("\n");

    GenSet *intStringIntersect = intersectSet(intSet, stringSet); // Get
intersection of integer set and string set

    // Check if result is NULL
    if(intStringIntersect != NULL) {
        printf("\nIntersection of Integer Set and String Set: ");
        displaySet(*intStringIntersect);
        deinitSet(intStringIntersect);
    } else {
        printf("Error getting intersectSet result.\n");
    }

    printf("\n");

    GenSet *stringSetIntersection = intersectSet(stringSet, stringSet2); //
Get intersection of string set and string set 2

    // Check if result is NULL
    if(stringSetIntersection != NULL) {
        printf("\nIntersection of String Set and String Set 2: ");
        displaySet(*stringSetIntersection);
        deinitSet(stringSetIntersection);
    } else {
        printf("Error getting intersectSet result.\n");
    }

    printf("\n");

    GenSet *intSetIntersection = intersectSet(intSet, intSet2); // Get
intersection of integer set and integer set 2

    // Check if result is NULL
    if(intSetIntersection != NULL) {
        printf("\nIntersection of Int Set and Int Set 2: ");
        displaySet(*intSetIntersection);
        deinitSet(intSetIntersection);
    } else {
        printf("Error getting intersectSet result.\n");
    }

    printf("\n");

    GenSet *intStringDiff = diffSet(intSet, stringSet); // Get difference
between integer set and string set

    // Check if result is NULL
    if(intStringDiff != NULL) {
        printf("\nDifference between Integer Set and String Set: ");
        displaySet(*intStringDiff);
        deinitSet(intStringDiff);
    } else {
        printf("Error getting diffSet result.\n");
    }

    deinitSet(&intSet); // Deinitialize integer set
    deinitSet(&stringSet); // Deinitialize string set
    deinitSet(&stringSet2); // Deinitialize string set 2
    deinitSet(&intSet2); // Deinitialize integer set 2
}
```

<u>SetMethodsA.c</u>

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "setMethodsA.h"

/**
 * Initializes a set with the given element type
 */
void initSet(GenSet *set, int elementType) {
    set->elementType = elementType;
    set->size = 0;
    set->elements = (void **)malloc(MAX_SET_SIZE * sizeof(void *)); //
Allocate memory for the array of elements
    if (set->elements == NULL) {
        printf("Memory allocation failed for elements in initSet\n");
        printf("\n");
        exit(EXIT_FAILURE);
    }
}

/**
 * Deinitializes a set
 */
void deinitSet(GenSet *set) {
    for (int i = 0; i < set->size; ++i) {
        free(set->elements[i]); // Free the memory allocated for each
element
    }
    free(set->elements); // Free the memory allocated for the array of
elements
    set->size = 0;
}

/**
 * Adds an element to the set
 */
int addToSet(GenSet *set, void *element) {
    if (set->size == MAX_SET_SIZE) {
        printf("Set is full\n");
        printf("\n");
        return 0;
    } else if (isSubsetSet(*set, element) != 0) {
        if(isalpha(*((char *)(element)))) {
            printf("Element '%s' already exists in set.\n", (char
*)(element));
        } else {
            printf("Element '%d' already exists in set.\n", *(int
*)(element));
        }
        return 0;
    } else if (set->elementType == 0) { // Integer type
        int *newElement = (int *)malloc(sizeof(int)); // Allocate memory
for the new element
        if (newElement == NULL) {
            printf("Memory allocation failed for newElement in
addToSet\n");
            printf("\n");
            exit(EXIT_FAILURE);
        }
        *newElement = *((int *)element); // Copy the value of the element
to the new element
        set->elements[set->size] = newElement; // Add the new element to
the set
    } else if (set->elementType == 1) { // String type
        char *newElement = (char *)malloc(MAX_STRING_LENGTH *
sizeof(char)); // Allocate memory for the new element
        if (newElement == NULL) {
            printf("Memory allocation failed for newElement in
addToSet\n");
            printf("\n");
            exit(EXIT_FAILURE);
        }
        strcpy(newElement, (char *)element); // Copy the value of the
element to the new element
        set->elements[set->size] = newElement; // Add the new element to
the set
```

```
    }
    set->size++;
    return 1;
}

/**
 * Displays the set
 */
void displaySet(GenSet set) {
    for (int i = 0; i < set.size; ++i) {
        // Check if element is of type string
        if(isalpha(*((char *)(set.elements[i])))) {
            printf("%s ", (char *)(set.elements[i]));
        } else {
            printf("%d ", * (int *)(set.elements[i]));
        }
    }
    printf("\n");
}

/**
 * Unions two sets if they are of the same type
 */
GenSet *unionSet(GenSet set1, GenSet set2) {
    // Check if sets are of the same type
    if (set1.elementType != set2.elementType) {
        printf("\nCannot perform union on sets of different types.\n");
        return NULL;
    }

    GenSet *unionSet = (GenSet*)malloc(sizeof(GenSet)); // Allocate memory
for the union set

    if (unionSet == NULL) {
        printf("Memory allocation failed for result in intersectSet\n");
        return NULL;
    }

    initSet(unionSet, set2.elementType);

    // Add elements of set1 to unionSet
    for (int i = 0; i < set1.size; ++i) {
        addToSet(unionSet, set1.elements[i]);
    }
    // Add elements of set2 to unionSet
    for (int i = 0; i < set2.size; ++i) {
        addToSet(unionSet, set2.elements[i]);
    }
    return unionSet;
}

/**
 * Intersects two sets if they are of the same type
 */
GenSet *intersectSet(GenSet set1, GenSet set2) {
    // Check if sets are of the same type
    if (set1.elementType != set2.elementType) {
        printf("Cannot perform union on sets of different types.\n");
        return NULL;
    }
    GenSet *result = (GenSet*)malloc(sizeof(GenSet)); // Allocate memory
for the intersection set

    if (result == NULL) {
        printf("Memory allocation failed for result in intersectSet\n");
        return NULL;
    }

    initSet(result, set1.elementType);

    // Add elements of set1 to result if they are also in set2
    for (int i = 0; i < set1.size; ++i) {
        if (isSubsetSet(set2, set1.elements[i])) {
            addToSet(result, set1.elements[i]);
        }
    }

    if (countSet(*result) == 0) {
        printf("No intersection found.");
    }

    return result;
}

/**
 * Finds the difference between two sets if they are of the same type
 */
GenSet *diffSet(GenSet set1, GenSet set2) {
    // Check if sets are of the same type
    if (set1.elementType != set2.elementType) {
        printf("Cannot perform union on sets of different types.\n");
        return NULL;
    }

    GenSet *diffSet = (GenSet*)malloc(sizeof(GenSet)); // Allocate memory
for the difference set

    if (diffSet == NULL) {
        printf("Memory allocation failed for result in intersectSet\n");
        return NULL;
    }

    initSet(diffSet, set1.elementType);

    // Add elements of set1 to result if they are not in set2
    for (int i = 0; i < set1.size; ++i) {
        int isFound = 0;
        for (int j = 0; j < set2.size; ++j) {
            if (set1.elementType == 0) { // Integer type
                if (*((int *)(set1.elements[i])) == *((int
*)(set2.elements[j]))) {
                    isFound = 1;
                    break;
                }
            } else if (set1.elementType == 1) { // String type
                if (strcmp((char *)(set1.elements[i]), (char
*)(set2.elements[j])) == 0) {
                    isFound = 1;
                    break;
                }
            }
        }
        if (!isFound) {
            addToSet(diffSet, set1.elements[i]);
        }
    }
    return diffSet;
}

/**
 * Counts the number of elements in the set
 */
int countSet(GenSet set) {
    return set.size;
}

/**
 * Checks if the given element is a subset of the set
 */
void *isSubsetSet(GenSet set1, void *element) {
    for (int i = 0; i < set1.size; ++i) {
        if (set1.elementType == 0) {
            if (*((int *)element) == *((int *)(set1.elements[i]))) {
                return set1.elements[i];
            }
        } else if (set1.elementType == 1) {
            if (strcmp((char *)element, (char *)(set1.elements[i])) == 0) {
                return set1.elements[i];
            }
        }
    }
    return 0;
}

/**
 * Checks if the set is empty
 */
int isEmptySet(GenSet set) {
    return set.size == 0;
}
```

In 'SetMethodsA.c' we find all the defined functions which can be used by the user.

In *'Figure 4.01'*, we see the '#include' tag which includes the needed libraries for the program to function and the header file needed for the struct to work and to make sure all the functions are defined correctly.

```
- Figure 4.01
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "setMethodsA.h"
```

In *'Figure 4.02'*, we can see the initSet function which initialises the set-in order to allocate memory space to that set. If the initialisation fails, meaning that there is no memory which the program can utilise, the program will output an error message and exit. The size of the memory is set to the maximum allowed set size which is 100.

```
- Figure 4.02
void initSet(GenSet *set, int elementType) {
    set->elementType = elementType;
    set->size = 0;
    set->elements = (void **)malloc(MAX_SET_SIZE * sizeof(void *));
    if (set->elements == NULL) {
        printf("Memory allocation failed for elements in initSet\n");
        printf("\n");
        exit(EXIT_FAILURE);
    }
}
```

In *'Figure 4.03'*, we can see the deinitSet function, which deinitialises the set, meaning frees up the memory taken by the set, to ensure there are no memory leaks. First it proceeds to free all the elements and then sets the set size to 0.

```
- Figure 4.03
void deinitSet(GenSet *set) {
    for (int i = 0; i < set->size; ++i) {
        free(set->elements[i]); // Free the memory allocated for each
element
    }
    free(set->elements); // Free the memory allocated for the array of
elements
    set->size = 0;
}
```

In *'Figure 4.04'*, we can see the addToSet function, which adds elements to a set.

It first checks if the set size is equal to the max set size and if so, it will output an error that the set is full and will not enter the element into the set. If this is found to not be the case it checks if the element is already in the set, and if the function isSubsetSet returns 1, meaning that there is already an element with the same value, the element will not be added, and an error is outputted to the user.

If both these checks are passed, the program then proceeds to input the element into the set. It first checks if the elementType is set to:

- 0 – representing an Integer. Here the program allocates memory for the element to be placed into, and then the memory address is placed into the elements.
- 1 – representing strings. Here the process is almost the same as the integer adding, but instead strcpy is used to copy the value inside of the element parameter into the current allocated memory and is then placed into the set elements.

After the elements are added the size of the set is increased by 1, and the program returns 1 to indicate that the adding of the element is successful.

```
- Figure 4.04
```

```
int addToSet(GenSet *set, void *element) {
    if (set->size == MAX_SET_SIZE) {
        printf("Set is full\n");
        printf("\n");
        return 0;
    } else if (isSubsetSet(*set, element) != 0) {
        if(isalpha(*((char *)(element)))) {
            printf("Element '%s' already exists in set.\n", (char
*)(element));
        } else {
            printf("Element '%d' already exists in set.\n", *(int
*)(element));
        }
        return 0;
    } else if (set->elementType == 0) { // Integer type
        int *newElement = (int *)malloc(sizeof(int));
        if (newElement == NULL) {
            printf("Memory allocation failed for newElement in
addToSet\n");
            printf("\n");
            exit(EXIT_FAILURE);
        }
        *newElement = *((int *)element);
        set->elements[set->size] = newElement;
    } else if (set->elementType == 1) { // String type
        char *newElement = (char *)malloc(MAX_STRING_LENGTH *
sizeof(char));
        if (newElement == NULL) {
            printf("Memory allocation failed for newElement in
addToSet\n");
            printf("\n");
            exit(EXIT_FAILURE);
        }
        strcpy(newElement, (char *)element);
        set->elements[set->size] = newElement;
    }
    set->size++;
    return 1;
}
```

In *'Figure 4.05'*, we can see the displaySet function, which outputs the elements in a set. This is done by first utilising the for loop make sure every element is outputted, but in the for loop we have an if condition to check whether the element being outputted is a string or an integer by utilising the isalpha function and outputting as intended.

```
- Figure 4.05
void displaySet(GenSet set) {
    for (int i = 0; i < set.size; ++i) {
        // Check if element is of type string
        if(isalpha(*((char *)(set.elements[i])))) {
            printf("%s ", (char *)(set.elements[i]));
        } else {
            printf("%d ", * (int *)(set.elements[i]));
        }
    }
    printf("\n");
}
```

In *'Figure 4.06'*, we have the unionSet function, which unionises two sets of the same type. First, we have the check of verifying that 2 sets are the same and if they are not of the same type, the user will get an error message and the union is not performed. A chunk of memory is then allocated for the union to take place. If the memory is not allocated, the user will get an error.

If memory is allocated, the set is then initialised using the initSet function and then both sets are added one after the other using 2 separate for loops with each calling the addToSet function to add the elements into the union set, first set 1 is put into the union set and then set 2 follows suit.

```
- Figure 4.06
GenSet *unionSet(GenSet set1, GenSet set2) {
    // Check if sets are of the same type
    if (set1.elementType != set2.elementType) {
        printf("\nCannot perform union on sets of different types.\n");
        return NULL;
    }

    GenSet *unionSet = (GenSet*)malloc(sizeof(GenSet)); // Allocate memory
for the union set

    if (unionSet == NULL) {
        printf("Memory allocation failed for result in intersectSet\n");
        return NULL;
    }

    initSet(unionSet, set2.elementType);

    // Add elements of set1 to unionSet
    for (int i = 0; i < set1.size; ++i) {
        addToSet(unionSet, set1.elements[i]);
    }
    // Add elements of set2 to unionSet
    for (int i = 0; i < set2.size; ++i) {
        addToSet(unionSet, set2.elements[i]);
    }
    return unionSet;
}
```

In *'Figure 4.07'*, we can see the intersectSet function, which intersects two sets of the same type. Here we again have the check to see if the sets are of the same type, after we try to allocate memory, if this is unsuccessful an error is given to the user.

If memory allocation is successful, we initialise the intersect set and we have a for loop to check if any of the elements of the 2nd set are the same as any elements in the 1st set and if they are they are added to the intersect set using the addToSet function. If there are no intersections an error is given telling the user.

```
- Figure 4.07
GenSet *intersectSet(GenSet set1, GenSet set2) {
    // Check if sets are of the same type
    if (set1.elementType != set2.elementType) {
        printf("Cannot perform union on sets of different types.\n");
        return NULL;
    }

    GenSet *result = (GenSet*)malloc(sizeof(GenSet)); // Allocate memory
for the intersection set

    if (result == NULL) {
        printf("Memory allocation failed for result in intersectSet\n");
        return NULL;
    }

    initSet(result, set1.elementType);

    // Add elements of set1 to result if they are also in set2
    for (int i = 0; i < set1.size; ++i) {
        if (isSubsetSet(set2, set1.elements[i])) {
            addToSet(result, set1.elements[i]);
        }
    }

    if (countSet(*result) == 0) {
        printf("No intersection found.");
    }

    return result;
}
```

In *'Figure 4.08'*, we have the diffSet function, which adds the different elements between 2 sets into a set called diffSet. First, we check if the 2 sets given by the user are of the same input type. After we try to allocate memory to the set and if that is successful we have 2 for loops and they check if the elements match or no, if they do they will be flagged and will be added using the addToSet function.

```
- Figure 4.08
GenSet *diffSet(GenSet set1, GenSet set2) {
    // Check if sets are of the same type
    if (set1.elementType != set2.elementType) {
        printf("Cannot perform union on sets of different types.\n");
        return NULL;
    }

    GenSet *diffSet = (GenSet*)malloc(sizeof(GenSet)); // Allocate memory
for the difference set

    if (diffSet == NULL) {
        printf("Memory allocation failed for result in intersectSet\n");
        return NULL;
    }

    initSet(diffSet, set1.elementType);

    // Add elements of set1 to result if they are not in set2
    for (int i = 0; i < set1.size; ++i) {
        int isFound = 0;
        for (int j = 0; j < set2.size; ++j) {
            if (set1.elementType == 0) { // Integer type
                if (*((int *)(set1.elements[i])) == *((int
*)(set2.elements[j]))) {
                    isFound = 1;
                    break;
                }
            } else if (set1.elementType == 1) { // String type
                if (strcmp((char *)(set1.elements[i]), (char
*)(set2.elements[j])) == 0) {
                    isFound = 1;
                    break;
                }
            }
        }
        if (!isFound) {
            addToSet(diffSet, set1.elements[i]);
        }
    }
    return diffSet;
}
```

In *'Figure 4.09'*, we have the countSet function, which returns the size of the set.

```
- Figure 4.09
int countSet(GenSet set) {
    return set.size;
}
```

In *'Figure 4.10'*, we have the isSubsetSet function, which checks if an element is a contained within the set, if yes it will return the element, if none of the elements are contained within the set it will return 0.

```
- Figure 4.10
void *isSubsetSet(GenSet set1, void *element) {
    for (int i = 0; i < set1.size; ++i) {
        if (set1.elementType == 0) {
            if (*((int *)element) == *((int *)(set1.elements[i]))) {
                return set1.elements[i];
            }
        } else if (set1.elementType == 1) {
            if (strcmp((char *)element, (char *)(set1.elements[i])) == 0) {
                return set1.elements[i];
            }
        }
    }
    return 0;
}
```

In *'Figure 4.11'*, we have the isEmptySet Function, which checks if a set is empty by returning the set size if it is equal to 0.

```
- Figure 4.11
int isEmptySet(GenSet set) {
    return set.size == 0;
}
```

setMethodsA.h

```
#ifndef ASSIGNMENT_QUESTION_2_SETMETHODSA_H
#define ASSIGNMENT_QUESTION_2_SETMETHODSA_H

#define MAX_STRING_LENGTH 64 // Maximum length of a string
#define MAX_SET_SIZE 100 // Maximum number of elements in a set

typedef struct {
    int elementType;
    void **elements;
    int size;
} GenSet; // Generic set

void initSet(GenSet *set, int elementType); // Initializes a set
void deinitSet(GenSet *set); // Deinitializes a set
int addToSet(GenSet *set, void *element); // Adds an element to the set
void displaySet(GenSet set); // Displays the set
GenSet *unionSet(GenSet set1, GenSet set2); // Returns the union of two
sets
GenSet *intersectSet(GenSet set1, GenSet set2); // Returns the intersection
of two sets
GenSet *diffSet(GenSet set1, GenSet set2);  // Returns the difference of
two sets
int countSet(GenSet set); // Returns the number of elements in the set
void *isSubsetSet(GenSet set1, void *element); // Checks if an element is a
subset of a set
int isEmptySet(GenSet set); // Checks if the set is empty

#endif //ASSIGNMENT_QUESTION_2_SETMETHODSA_H
```

In *‘Figure 5.1’*, we can see the ‘#define’ tags which includes predefined value for the max string length and the max set size.

```
- Figure 5.1
#define MAX_STRING_LENGTH 64 // Maximum length of a string
#define MAX_SET_SIZE 100 // Maximum number of elements in a set
```

In *‘Figure 5.2’*, the struct is defined with its respective variables which are:

- elementType – which represent an integer representing the type of set it is if it is an integer or string set.
- Elements – which contains the elements within the set as a pointer value.
- Size – this contains the number of elements in the set.

After the struct, we find all the defined functions to be created in the .c file, these functions are:

- initSet
- deinitSet
- addToSet
- displaySet
- unionSet
- intersectSet
- diffSet
- countSet

- isSubsetSet
- isEmptySet

```
- Figure 5.2
typedef struct {
    int elementType;
    void **elements;
    int size;
} GenSet; // Generic set

void initSet(GenSet *set, int elementType); // Initializes a set
void deinitSet(GenSet *set); // Deinitializes a set
int addToSet(GenSet *set, void *element); // Adds an element to the set
void displaySet(GenSet set); // Displays the set
GenSet *unionSet(GenSet set1, GenSet set2); // Returns the union of two
sets
GenSet *intersectSet(GenSet set1, GenSet set2); // Returns the intersection
of two sets
GenSet *diffSet(GenSet set1, GenSet set2);  // Returns the difference of
two sets
int countSet(GenSet set); // Returns the number of elements in the set
void *isSubsetSet(GenSet set1, void *element); // Checks if an element is a
subset of a set
int isEmptySet(GenSet set); // Checks if the set is empty
```

Question 2B:

Below is all the code used to make Question 2B and some explanations of the code.

Question2B.c

```
#include <stdio.h>
#include <stdlib.h>
#include "setMethodsB.h"
#include "testingStructFileB.h"

int main() {
    // Removes files from previous runs
    remove("planeSet.txt");
    remove("planetSet.txt");

    // Clears buffer in cases where terminal does not output anything
    setbuf(stdout, 0);

    printf("Welcome to the Set Operations Program!\n\n");

    // Creates a set of Planes
    GenSet *planeSet1 = ((GenSet *) malloc(sizeof(GenSet)));

    // Initialises set with the following parameters:
    initSet(planeSet1,1, 20, &getSizeOfPlanes, &comparePlanes,
&addPlanesToSet, &displayPlanes, &exportPlanes);

    // Adds the following elements to the set:
    addToSet(planeSet1, &(Planes){"Boeing 747", 416});
    addToSet(planeSet1, &(Planes){"Boeing 777", 300});
    addToSet(planeSet1, &(Planes){"Boeing 747", 416}); // Duplicate element

    // Displays the set
    displaySet(planeSet1);

    // Creates a set of Planets
    GenSet *planetSet1 = ((GenSet *) malloc(sizeof(GenSet)));

    // Initialises set with the following parameters:
    initSet(planetSet1,2, 20, &getSizeOfPlanets, &comparePlanets,
&addPlanetsToSet, &displayPlanets, &exportPlanets);

    // Adds the following elements to the set:
    addToSet(planetSet1, &(Planets){"Earth", 40075});
    addToSet(planetSet1, &(Planets){"Jupiter", 439264});

    // Displays the set
    displaySet(planetSet1);

    GenSet *planeSet2 = ((GenSet *) malloc(sizeof(GenSet)));

    // Initialises set with the following parameters:
    initSet(planeSet2,1, 20, &getSizeOfPlanes, &comparePlanes,
&addPlanesToSet, &displayPlanes, &exportPlanes);

    // Adds the following elements to the set:
    addToSet(planeSet2, &(Planes){"Boeing 747", 416});

    // Displays the set
    displaySet(planeSet2);

    // Performs union
    GenSet *unionSet1 = unionSet(planeSet1, planetSet1);

    // Displays the union set
    if (unionSet1 != NULL) {
        printf("\nUnion of Plane Set and Planet Set:\n");
        displaySet(unionSet1);
        deinitSet(unionSet1); // Deinitialises the set
    }

    // Performs intersection
    GenSet *intersectSet1 = intersectSet(planeSet1, planeSet2);

    // Displays the intersection set
    if (intersectSet1 != NULL) {
        printf("\nIntersection of Plane Set 1 and Plane Set 2:\n");
        displaySet(intersectSet1);
        deinitSet(intersectSet1); // Deinitialises the set
    }

    // Performs difference
    GenSet *diffSet1 = diffSet(planeSet1, planetSet1);

    // Displays the difference set
    if (diffSet1 != NULL) {
        printf("\nDifference of Plane Set and Planet Set:\n");
        displaySet(diffSet1);
        deinitSet(diffSet1); // Deinitialises the set
    }
    // Counts the number of elements in the plane set
    printf("\nNumber of elements in Plane Set: %d\n", countSet(planeSet1));

    // Counts the number of elements in the planet set
    printf("\nNumber of elements in Planet Set: %d\n",
countSet(planetSet1));

    // Checks if the element Earth is a subset of the planet set
    Planets *planet = (Planets *)isSubsetSet(planetSet1,
&(Planets){"Earth", 40075});

    // Displays the element if it is a subset
    if (planet != NULL) {
        printf("\nPlanet found in Planet Set:\n");
        displayPlanets(planet);
    } else {
        printf("\n'Earth' not found in Planet Set\n");
    }

    // Checks if the plane set is empty
    printf("\nIs Plane Set empty? %s\n", isEmptySet(planeSet1) ? "Yes" :
"No");

    // Checks if the planet set is empty
    printf("\nIs Planet Set empty? %s\n", isEmptySet(planetSet1) ? "Yes" :
"No");

    // Exports the plane set and planet set to planeSet.txt and
planetSet.txt respectively
    printf("\nExporting Plane Set to planeSet.txt and Planet Set to
planetSet.txt...\n");

    // Calls the export function
    export(planeSet1, "planeSet.txt");
    export(planetSet1, "planetSet.txt");

    // Deinitialises the sets
    deinitSet(planeSet1);
    deinitSet(planetSet1);

    printf("\nThank you for using this program. Goodbye!\n");

    return 0;
}
```

The above code is explained in separate snippets below:

In *'Figure 6.1'*, we first start with the '#include' tags which include predefined C header libraries and 2 header files which we have created. After we start with the main method. In here we first start by deleting any previous text files created, then we go to initialise the plane set and add some elements to it. After we display that set, and we go to initialise the planet set, add some elements to it and display it. After we add a second plane set with 1 element so that we can use later during our testing.

```
- Figure 6.1
int main() {
    // Removes files from previous runs
    remove("planeSet.txt");
    remove("planetSet.txt");

    // Clears buffer in cases where terminal does not output anything
    setbuf(stdout, 0);

    printf("Welcome to the Set Operations Program!\n\n");

    // Creates a set of Planes
    GenSet *planeSet1 = ((GenSet *) malloc(sizeof(GenSet)));

    // Initialises set with the following parameters:
    initSet(planeSet1,1, 20, &getSizeOfPlanes, &comparePlanes,
&addPlanesToSet, &displayPlanes, &exportPlanes);

    // Adds the following elements to the set:
    addToSet(planeSet1, &(Planes){"Boeing 747", 416});
    addToSet(planeSet1, &(Planes){"Boeing 777", 300});
    addToSet(planeSet1, &(Planes){"Boeing 747", 416}); // Duplicate element

    // Displays the set
    displaySet(planeSet1);

    // Creates a set of Planets
    GenSet *planetSet1 = ((GenSet *) malloc(sizeof(GenSet)));

    // Initialises set with the following parameters:
    initSet(planetSet1,2, 20, &getSizeOfPlanets, &comparePlanets,
&addPlanetsToSet, &displayPlanets, &exportPlanets);

    // Adds the following elements to the set:
    addToSet(planetSet1, &(Planets){"Earth", 40075});
    addToSet(planetSet1, &(Planets){"Jupiter", 439264});

    // Displays the set
    displaySet(planetSet1);

    GenSet *planeSet2 = ((GenSet *) malloc(sizeof(GenSet)));

    // Initialises set with the following parameters:
    initSet(planeSet2,1, 20, &getSizeOfPlanes, &comparePlanes,
&addPlanesToSet, &displayPlanes, &exportPlanes);

    // Adds the following elements to the set:
    addToSet(planeSet2, &(Planes){"Boeing 747", 416});

    // Displays the set
    displaySet(planeSet2);
```

In *'Figure 6.2'*, we see the program work out the union, intersect and difference of two sets, these are then outputted accordingly. Here we also see the countSet being used by counting the number of elements in the plane set and the planet set. We also use the isSubsetSet function to check if "Earth" is a subset of the planet set.

```
- Figure 6.2
// Performs union
GenSet *unionSet1 = unionSet(planeSet1, planetSet1);

// Displays the union set
```

```
if (unionSet1 != NULL) {
    printf("\nUnion of Plane Set and Planet Set:\n");
    displaySet(unionSet1);
    deinitSet(unionSet1); // Deinitialises the set
}

// Performs intersection
GenSet *intersectSet1 = intersectSet(planeSet1, planeSet2);

// Displays the intersection set
if (intersectSet1 != NULL) {
    printf("\nIntersection of Plane Set 1 and Plane Set 2:\n");
    displaySet(intersectSet1);
    deinitSet(intersectSet1); // Deinitialises the set
}

// Performs difference
GenSet *diffSet1 = diffSet(planeSet1, planetSet1);

// Displays the difference set
if (diffSet1 != NULL) {
    printf("\nDifference of Plane Set and Planet Set:\n");
    displaySet(diffSet1);
    deinitSet(diffSet1); // Deinitialises the set
}

// Counts the number of elements in the plane set
printf("\nNumber of elements in Plane Set: %d\n", countSet(planeSet1));

// Counts the number of elements in the planet set
printf("\nNumber of elements in Planet Set: %d\n", countSet(planetSet1));

// Checks if the element Earth is a subset of the planet set
Planets *planet = (Planets *)isSubsetSet(planetSet1, &(Planets){"Earth",
40075});

// Displays the element if it is a subset
if (planet != NULL) {
    printf("\nPlanet found in Planet Set:\n");
    displayPlanets(planet);
} else {
    printf("\n'Earth' not found in Planet Set\n");
}
```

In *'Figure 6.3'*, the program runs some more tests to ensure all the functions are working as intended. It tests the isEmptySet function to check if sets are empty and it exports the plane set and planet set into their respective text files. When done the sets are deinitialised from the memory.

```
- Figure 6.3
// Checks if the plane set is empty
printf("\nIs Plane Set empty? %s\n", isEmptySet(planeSet1) ? "Yes" : "No");

// Checks if the planet set is empty
printf("\nIs Planet Set empty? %s\n", isEmptySet(planetSet1) ? "Yes" :
"No");

// Exports the plane set and planet set to planeSet.txt and planetSet.txt
respectively
```

```
printf("\nExporting Plane Set to planeSet.txt and Planet Set to
planetSet.txt...\n");

// Calls the export function
export(planeSet1, "planeSet.txt");
export(planetSet1, "planetSet.txt");

// Deinitialises the sets
deinitSet(planeSet1);
deinitSet(planetSet1);
```

setMethodsB.c

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "setMethodsB.h"

/**
  * This function initialises the set with the following parameters
 */
void initSet(GenSet *set, int elementType, int initSize, getSizeOfFP
getSizeOfFP, compareFP compareFP, addToSetFP addToSetFP, displayFP
displayFP, exportFP exportFP){
    set->FunctionPointers.getSizeOfFP = getSizeOfFP; // getSizeOfFP is a
function pointer
    set->FunctionPointers.compareFP = compareFP; // compareFP is a function
pointer
    set->FunctionPointers.addToSetFP = addToSetFP; // addToSetFP is a
function pointer
    set->FunctionPointers.displayFP = displayFP; // displayFP is a function
pointer
    set->FunctionPointers.exportFP = exportFP; // exportFP is a function
pointer
    set->elementType = elementType; // elementType is an integer
    set->memorySize = initSize; // initSize is an integer
    set->usedSize = 0; // usedSize is an integer
    set->elements = (void *)malloc( initSize * (*set-
>FunctionPointers.getSizeOfFP)()); // getSizeOfFP is a function pointer

    // Check if memory allocation failed
    if (set->elements == NULL) {
        printf("Memory allocation failed for elements or set of sets in
initSet\n");
        exit(EXIT_FAILURE);
    }
}

/**
  * This function deinitialises the set
 */
void deinitSet(GenSet *set) {
    free(set->elements); // free elements
    free(set); // free the set
}
```

```c
/**
  * This function adds an element to the set
 */
int addToSet(GenSet *set, void *element) {
    // Check if set is full
    if (set -> usedSize == set -> memorySize) {
        printf("Set is full, adding more memory...\n");
        if((set->elements = (void *)realloc(set->elements, (set->
memorySize + 5) * (*set->FunctionPointers.getSizeOfFP)())) == NULL){
            printf("Memory allocation failed for elements or set of sets in
addToSet\n");
            exit(EXIT_FAILURE);
        }
        set-> memorySize += 5;
    }

    // Check for duplicates
    for (int i = 0; i < set->usedSize; i++) {
        if ((*set->FunctionPointers.compareFP)((&((char *)set->elements)[i
* (*set->FunctionPointers.getSizeOfFP)()]), element) == 1) {
            if(isalpha(*(char *)(element))) {
                printf("\nElement '%s' already exists in set.\n\n", (char
*)(element));
            } else {
                printf("\nElement '%d' already exists in set.\n\n", *(int
*)(element));
            }
            return 0;
        }
    }

    // Add element to set
    if ((*set->FunctionPointers.addToSetFP)((&((char *)set->elements)[set-
>usedSize * (*set->FunctionPointers.getSizeOfFP)()]), element) == 0) {
        printf("Unable to add element %s to set\n", (char *)element);
        return 0;
    } else {

        if (set->elementType == 1 || set->elementType == 2) {
            if (isalpha(*((char *) (element)))) {
                printf("Element '%s' added to set\n", (char *) element);
            } else {
                printf("Element '%d' added to set\n", *(int *) (element));
            }
        }
    }

    set->usedSize++;

    return 1;
}

/**
  * This function displays the set
 */
void displaySet(GenSet *set) {
    for (int i = 0; i < set->usedSize; i++) {
        (*set->FunctionPointers.displayFP)(&((char *)set->elements)[i *
(*set->FunctionPointers.getSizeOfFP)()]);
    }
    printf("\n");
```

```
}

/**
  * This function performs union
 */
GenSet *unionSet(GenSet *set1, GenSet *set2) {
    // Check if the sets have different element types
    if (set1->elementType != set2->elementType) {
        printf("Error in Union: The sets have different element types\n");
        return NULL; // Return NULL if the sets have different element
types
    }

    GenSet *result = (GenSet *)malloc(sizeof(GenSet));

    if (result == NULL) {
        printf("Memory allocation failed for result in setUnion\n");
        return NULL;
    }

    initSet(result, 3, set1->memorySize + set2->memorySize, set1-
>FunctionPointers.getSizeOfFP, set1->FunctionPointers.compareFP, set1-
>FunctionPointers.addToSetFP, set1->FunctionPointers.displayFP, set1-
>FunctionPointers.exportFP);

    for (int i = 0; i < set1->usedSize; i++) {
        addToSet(result, &((char *)set1->elements)[i * (*set1-
>FunctionPointers.getSizeOfFP)()]);
    }

    for (int i = 0; i < set2->usedSize; i++) {
        addToSet(result, &((char *)set2->elements)[i * (*set2-
>FunctionPointers.getSizeOfFP)()]);
    }

    return result;
}

/**
  * This function performs intersection
 */
GenSet *intersectSet(GenSet *set1, GenSet *set2) {
    // Check if the sets have different element types
    if (set1->elementType != set2->elementType) {
        printf("Error in Intersection: The sets have different element
types\n");
        return NULL; // Return NULL if the sets have different element
types
    }

    GenSet *result = (GenSet *)malloc(sizeof(GenSet));

    if (result == NULL) {
        printf("Memory allocation failed for result in setIntersection\n");
        return NULL;
    }

    initSet(result, 4, set1->memorySize, set1-
>FunctionPointers.getSizeOfFP, set1->FunctionPointers.compareFP, set1-
>FunctionPointers.addToSetFP, set1->FunctionPointers.displayFP, set1-
>FunctionPointers.exportFP);
```

```
    for (int i = 0; i < set1->usedSize; i++) {
        for (int j = 0; j < set2->usedSize; j++) {
            if ((*set1->FunctionPointers.compareFP)(&((char *)set1-
>elements)[i * (*set1->FunctionPointers.getSizeOfFP)()], &((char *)set2-
>elements)[j * (*set2->FunctionPointers.getSizeOfFP)()]) == 0) {
                addToSet(result, &((char *)set1->elements)[i * (*set1-
>FunctionPointers.getSizeOfFP)()]);
            }
        }
    }

    return result;
}

/**
  * This function performs difference
 */
GenSet *diffSet(GenSet *set1, GenSet *set2) {
    // Check if the sets have different element types
    if (set1->elementType != set2->elementType) {
        printf("Error in Difference: The sets have different element
types\n");
        return NULL; // Return NULL if the sets have different element
types
    }

    GenSet *result = (GenSet *)malloc(sizeof(GenSet));

    if (result == NULL) {
        printf("Memory allocation failed for result in setDifference\n");
        return NULL;
    }

    initSet(result, 5, set1->memorySize, set1-
>FunctionPointers.getSizeOfFP, set1->FunctionPointers.compareFP, set1-
>FunctionPointers.addToSetFP, set1->FunctionPointers.displayFP, set1-
>FunctionPointers.exportFP);

    for (int i = 0; i < set1->usedSize; i++) {
        int found = 0;
        for (int j = 0; j < set2->usedSize; j++) {
            if ((*set1->FunctionPointers.compareFP)(&((char *)set1-
>elements)[i * (*set1->FunctionPointers.getSizeOfFP)()], &((char *)set2-
>elements)[j * (*set2->FunctionPointers.getSizeOfFP)()]) == 0) {
                found = 1;
                break;
            }
        }
        if (!found) {
            addToSet(result, &((char *)set1->elements)[i * (*set1-
>FunctionPointers.getSizeOfFP)()]);
        }
    }

    return result;
}

/**
  * This function counts the number of elements in the set
 */
```

```
int countSet(GenSet *set) {
    return set->usedSize;
}

/**
  * This function checks if the element is a subset of the set
 */
void *isSubsetSet(GenSet *set, void *element) {
    for (int i = 0; i < set->usedSize; i++) {
        if ((*set->FunctionPointers.compareFP)(&((char *)set->elements)[i *
(*set->FunctionPointers.getSizeOfFP)()], element) == 1) {
            return &((char *)set->elements)[i * (*set-
>FunctionPointers.getSizeOfFP)()];
        }
    }
    return NULL;
}

/**
  * This function checks if the set is empty
 */
int isEmptySet(GenSet *set) {
    return set->usedSize == 0;
}

/**
  * This function exports the set to a file
 */
void export(GenSet *set, char *filename) {
    for (int i = 0; i < set->usedSize; i++) {
        (*set->FunctionPointers.exportFP)(&((char *)set->elements)[i *
(*set->FunctionPointers.getSizeOfFP)()], filename, i);
    }
}
```

Below are a few explanations of the above code:

In *'Figure 7.1'*, we can see that most of the code here is generally the same functions as defined in Question2A, but instead here almost all the code has been changed to utilise function pointers instead, but at heart the code has the same function as the one previous.

```
- Figure 7.1
/**
  * This function initialises the set with the following parameters
 */
void initSet(GenSet *set, int elementType, int initSize, getSizeOfFP
getSizeOfFP, compareFP compareFP, addToSetFP addToSetFP, displayFP
displayFP, exportFP exportFP){
    set->FunctionPointers.getSizeOfFP = getSizeOfFP; // getSizeOfFP is a
function pointer
    set->FunctionPointers.compareFP = compareFP; // compareFP is a function
pointer
    set->FunctionPointers.addToSetFP = addToSetFP; // addToSetFP is a
function pointer
    set->FunctionPointers.displayFP = displayFP; // displayFP is a function
pointer
    set->FunctionPointers.exportFP = exportFP; // exportFP is a function
pointer
    set->elementType = elementType; // elementType is an integer
    set->memorySize = initSize; // initSize is an integer
    set->usedSize = 0; // usedSize is an integer
    set->elements = (void *)malloc( initSize * (*set-
>FunctionPointers.getSizeOfFP)()); // getSizeOfFP is a function pointer

    // Check if memory allocation failed
    if (set->elements == NULL) {
        printf("Memory allocation failed for elements or set of sets in
initSet\n");
        exit(EXIT_FAILURE);
    }
}

/**
  * This function deinitialises the set
 */
void deinitSet(GenSet *set) {
    free(set->elements); // free elements
    free(set); // free the set
}

/**
  * This function adds an element to the set
 */
int addToSet(GenSet *set, void *element) {
    // Check if set is full
    if (set -> usedSize == set -> memorySize) {
        printf("Set is full, adding more memory...\n");
        if((set->elements = (void *)realloc(set->elements, (set->
memorySize + 5) * (*set->FunctionPointers.getSizeOfFP)())) == NULL){
            printf("Memory allocation failed for elements or set of sets in
addToSet\n");
            exit(EXIT_FAILURE);
        }
        set-> memorySize += 5;
    }

    // Check for duplicates
    for (int i = 0; i < set->usedSize; i++) {
        if ((*set->FunctionPointers.compareFP)((&((char *)set->elements)[i
* (*set->FunctionPointers.getSizeOfFP)()]), element) == 1) {
            if(isalpha(*(char *)(element))) {
                printf("\nElement '%s' already exists in set.\n\n", (char
*)(element));
            } else {
                printf("\nElement '%d' already exists in set.\n\n", *(int
*)(element));
            }
            return 0;
        }
    }

    // Add element to set
    if ((*set->FunctionPointers.addToSetFP)((&((char *)set->elements)[set-
>usedSize * (*set->FunctionPointers.getSizeOfFP)()]), element) == 0) {
        printf("Unable to add element %s to set\n", (char *)element);
        return 0;
    } else {

        if (set->elementType == 1 || set->elementType == 2) {
            if (isalpha(*((char *) (element)))) {
```

```c
                printf("Element '%s' added to set\n", (char *) element);
            } else {
                printf("Element '%d' added to set\n", *(int *) (element));
            }
        }
    }

    set->usedSize++;

    return 1;
}

/**
  * This function displays the set
 */
void displaySet(GenSet *set) {
    for (int i = 0; i < set->usedSize; i++) {
        (*set->FunctionPointers.displayFP)(&((char *)set->elements)[i *
(*set->FunctionPointers.getSizeOfFP)()]);
    }
    printf("\n");
}

/**
  * This function performs union
 */
GenSet *unionSet(GenSet *set1, GenSet *set2) {
    // Check if the sets have different element types
    if (set1->elementType != set2->elementType) {
        printf("Error in Union: The sets have different element types\n");
        return NULL; // Return NULL if the sets have different element
types
    }

    GenSet *result = (GenSet *)malloc(sizeof(GenSet));

    if (result == NULL) {
        printf("Memory allocation failed for result in setUnion\n");
        return NULL;
    }

    initSet(result, 3, set1->memorySize + set2->memorySize, set1-
>FunctionPointers.getSizeOfFP, set1->FunctionPointers.compareFP, set1-
>FunctionPointers.addToSetFP, set1->FunctionPointers.displayFP, set1-
>FunctionPointers.exportFP);

    for (int i = 0; i < set1->usedSize; i++) {
        addToSet(result, &((char *)set1->elements)[i * (*set1-
>FunctionPointers.getSizeOfFP)()]);
    }

    for (int i = 0; i < set2->usedSize; i++) {
        addToSet(result, &((char *)set2->elements)[i * (*set2-
>FunctionPointers.getSizeOfFP)()]);
    }

    return result;
}

/**
  * This function performs intersection
 */
GenSet *intersectSet(GenSet *set1, GenSet *set2) {
    // Check if the sets have different element types
    if (set1->elementType != set2->elementType) {
        printf("Error in Intersection: The sets have different element
types\n");
        return NULL; // Return NULL if the sets have different element
types
    }

    GenSet *result = (GenSet *)malloc(sizeof(GenSet));

    if (result == NULL) {
        printf("Memory allocation failed for result in setIntersection\n");
        return NULL;
    }

    initSet(result, 4, set1->memorySize, set1-
>FunctionPointers.getSizeOfFP, set1->FunctionPointers.compareFP, set1-
>FunctionPointers.addToSetFP, set1->FunctionPointers.displayFP, set1-
>FunctionPointers.exportFP);

    for (int i = 0; i < set1->usedSize; i++) {
        for (int j = 0; j < set2->usedSize; j++) {
            if ((*set1->FunctionPointers.compareFP)(&((char *)set1-
>elements)[i * (*set1->FunctionPointers.getSizeOfFP)()], &((char *)set2-
>elements)[j * (*set2->FunctionPointers.getSizeOfFP)()]) == 0) {
                addToSet(result, &((char *)set1->elements)[i * (*set1-
>FunctionPointers.getSizeOfFP)()]);
            }
        }
    }

    return result;
}

/**
  * This function performs difference
 */
GenSet *diffSet(GenSet *set1, GenSet *set2) {
    // Check if the sets have different element types
    if (set1->elementType != set2->elementType) {
        printf("Error in Difference: The sets have different element
types\n");
        return NULL; // Return NULL if the sets have different element
types
    }

    GenSet *result = (GenSet *)malloc(sizeof(GenSet));

    if (result == NULL) {
        printf("Memory allocation failed for result in setDifference\n");
        return NULL;
    }

    initSet(result, 5, set1->memorySize, set1-
>FunctionPointers.getSizeOfFP, set1->FunctionPointers.compareFP, set1-
>FunctionPointers.addToSetFP, set1->FunctionPointers.displayFP, set1-
>FunctionPointers.exportFP);

    for (int i = 0; i < set1->usedSize; i++) {
        int found = 0;
        for (int j = 0; j < set2->usedSize; j++) {
            if ((*set1->FunctionPointers.compareFP)(&((char *)set1-
>elements)[i * (*set1->FunctionPointers.getSizeOfFP)()], &((char *)set2-
>elements)[j * (*set2->FunctionPointers.getSizeOfFP)()]) == 0) {
                found = 1;
                break;
            }
        }
        if (!found) {
            addToSet(result, &((char *)set1->elements)[i * (*set1-
>FunctionPointers.getSizeOfFP)()]);
        }
    }

    return result;
}

/**
  * This function counts the number of elements in the set
 */
int countSet(GenSet *set) {
    return set->usedSize;
}

/**
  * This function checks if the element is a subset of the set
 */
void *isSubsetSet(GenSet *set, void *element) {
    for (int i = 0; i < set->usedSize; i++) {
        if ((*set->FunctionPointers.compareFP)(&((char *)set->elements)[i *
(*set->FunctionPointers.getSizeOfFP)()], element) == 1) {
            return &((char *)set->elements)[i * (*set-
>FunctionPointers.getSizeOfFP)()];
        }
    }
    return NULL;
}

/**
  * This function checks if the set is empty
 */
int isEmptySet(GenSet *set) {
    return set->usedSize == 0;
}
```

In *'Figure 7.2'*, we see a new function defined called export. Here all the elements in the given set are exported to a text file of the user's choice.

```
- Figure 7.2
void export(GenSet *set, char *filename) {
    for (int i = 0; i < set->usedSize; i++) {
        (*set->FunctionPointers.exportFP)(&((char *)set->elements)[i *
(*set->FunctionPointers.getSizeOfFP)()], filename, i);
    }
}
```

setMethodsB.h

```
#ifndef ASSIGNMENT_QUESTION_2_SETMETHODSB_H
#define ASSIGNMENT_QUESTION_2_SETMETHODSB_H

typedef size_t (*getSizeOfFP)(); // Function pointer to getSizeOf
typedef int (*compareFP)(void *element1, void *element2); // Function
pointer to compare
typedef int (*addToSetFP)(void *element1, void *element2); // Function
pointer to addToSet
typedef void (*displayFP)(void *element); // Function pointer to display
typedef void (*exportFP)(void *element, char *filename, int index); //
Function pointer to export

// Function pointer struct
typedef struct FP {
    getSizeOfFP getSizeOfFP;
    compareFP compareFP;
    addToSetFP addToSetFP;
    displayFP displayFP;
    exportFP exportFP;
} FP;

// Generic Set struct
typedef struct GenSet {
    int elementType;
    void *elements;
    int memorySize;
    int usedSize;
    FP FunctionPointers;
} GenSet;

void initSet(GenSet *set, int elementType, int initSize, getSizeOfFP
getSizeOfFP, compareFP compareFP, addToSetFP addToSetFP, displayFP
displayFP, exportFP exportFP); // Initialises the set
void deinitSet(GenSet *set); // Deinitialises the set
int addToSet(GenSet *set, void *element); // Adds an element to the set
void displaySet(GenSet *set); // Displays the set
GenSet *unionSet(GenSet *set1, GenSet *set2); // Performs union
GenSet *intersectSet(GenSet *set1, GenSet *set2); // Performs intersection
GenSet *diffSet(GenSet *set1, GenSet *set2); // Performs difference
int countSet(GenSet *set); // Counts the number of elements in the set
void *isSubsetSet(GenSet *set1, void *element); // Checks if the element is
a subset of the set
int isEmptySet(GenSet *set); // Checks if the set is empty
void export(GenSet *set, char *filename); // Exports the set to a file

#endif //ASSIGNMENT_QUESTION_2_SETMETHODSB_H
```

Below is an explanation of some of the above code:

In *'Figure 8.1'*, we can see the definition of function pointers, which can be used later for structs with different types. These function pointers include:

- getSize - which gets the number of elements.
- compare – compares 2 sets of the same type.

- addToSet – adds elements to a set.
- display – displays the set.
- export – exports the set to a text file.

```
- Figure 8.1
typedef size_t (*getSizeOfFP)(); // Function pointer to getSizeOf
typedef int (*compareFP)(void *element1, void *element2); // Function
pointer to compare
typedef int (*addToSetFP)(void *element1, void *element2); // Function
pointer to addToSet
typedef void (*displayFP)(void *element); // Function pointer to display
typedef void (*exportFP)(void *element, char *filename, int index); //
Function pointer to export
```

In *'Figure 8.2'*, we can see the structs of both Function Pointers and the GenSet, this was done to ensure that we have a struct for the operations, and a struct for the general structure of the set.

```
- Figure 8.2
// Function pointer struct
typedef struct FP {
    getSizeOfFP getSizeOfFP;
    compareFP compareFP;
    addToSetFP addToSetFP;
    displayFP displayFP;
    exportFP exportFP;
} FP;

// Generic Set struct
typedef struct GenSet {
    int elementType;
    void *elements;
    int memorySize;
    int usedSize;
    FP FunctionPointers;
} GenSet;
```

In *'Figure 8.3'*, one can see the definition of the functions to be used, which are:

- initSet
- deinitSet
- addToSet
- displaySet
- unionSet
- intersectSet
- diffSet
- countSet
- isSubsetSet
- isEmptySet
- Export

```
- Figure 8.3
```

```
void initSet(GenSet *set, int elementType, int initSize, getSizeOfFP
getSizeOfFP, compareFP compareFP, addToSetFP addToSetFP, displayFP
displayFP, exportFP exportFP); // Initialises the set
void deinitSet(GenSet *set); // Deinitialises the set
int addToSet(GenSet *set, void *element); // Adds an element to the set
void displaySet(GenSet *set); // Displays the set
GenSet *unionSet(GenSet *set1, GenSet *set2); // Performs union
GenSet *intersectSet(GenSet *set1, GenSet *set2); // Performs intersection
GenSet *diffSet(GenSet *set1, GenSet *set2); // Performs difference
int countSet(GenSet *set); // Counts the number of elements in the set
void *isSubsetSet(GenSet *set1, void *element); // Checks if the element is
a subset of the set
int isEmptySet(GenSet *set); // Checks if the set is empty
void export(GenSet *set, char *filename); // Exports the set to a file
```

testingStructFileB.h

```
#ifndef ASSIGNMENT_QUESTION_2_TESTINGSTRUCTFILEB_H
#define ASSIGNMENT_QUESTION_2_TESTINGSTRUCTFILEB_H

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

#define MAX_STRING_LENGTH 64

// Structs for Planes
typedef struct Planes {
    char name[MAX_STRING_LENGTH];
    int capacity;
} Planes;

// Structs for Planets
typedef struct Planets {
    char name[MAX_STRING_LENGTH];
    int circumference;
} Planets;

// Returns the size of Planes
size_t getSizeOfPlanes(){
    return sizeof(Planes);
}

// Returns the size of Planets
size_t getSizeOfPlanets(){
    return sizeof(Planets);
}

// Compares Plane sets
int comparePlanes(void *element1, void *element2) {
    Planes *plane1 = (Planes *)element1;
    Planes *plane2 = (Planes *)element2;

    if ((strcmp(plane1->name, plane2->name) == 0) && (plane1->capacity ==
plane2->capacity)) {
        return 1;
    } else {
        return 0;
    }
}

// Compares Planet sets
int comparePlanets(void *element1, void *element2) {
    Planets *planet1 = (Planets *)element1;
    Planets *planet2 = (Planets *)element2;

    if ((strcmp(planet1->name, planet2->name) == 0) && (planet1-
>circumference == planet2->circumference)) {
        return 1;
    } else {
        return 0;
    }
}

// Adds Plane sets to the set
int addPlanesToSet(void *element1, void *element2){
    Planes *plane1 = (Planes *)element1;
    Planes *plane2 = (Planes *)element2;

    strcpy(plane1->name, plane2->name);

    plane1->capacity += plane2->capacity;

    return 1;
}

// Adds Planet sets to the set
int addPlanetsToSet(void *element1, void *element2){
    Planets *planet1 = (Planets *)element1;
    Planets *planet2 = (Planets *)element2;

    strcpy(planet1->name, planet2->name);

    planet1->circumference += planet2->circumference;

    return 1;
}

// Displays Plane sets
void displayPlanes(void *element){
    Planes *plane = (Planes *)element;

    printf("Plane name: %s\n", plane->name);
    printf("Plane capacity: %d\n\n", plane->capacity);
}

// Displays Planet sets
void displayPlanets(void *element){
    Planets *planet = (Planets *)element;

    printf("Planet name: %s\n", planet->name);
    printf("Planet circumference: %d\n\n", planet->circumference);
}

// Exports Plane sets
void exportPlanes(void *element, char *filename, int index){
    Planes *plane = (Planes *)element;
    FILE *file = fopen(filename, "a");

    if (file == NULL){
        printf("Unable to open file %s\n", filename);
        return;
    }

    if (index == 0){
        fprintf(file, "Plane Set:\n\n");
    }

    fprintf(file, "%d.\tName: %s\n", index + 1, plane->name);
    fprintf(file, "\tCapacity: %d\n\n", plane->capacity);

    fclose(file);
}

// Exports Planet sets
void exportPlanets(void *element, char *filename, int index){
    Planets *planet = (Planets *)element;

    FILE *file = fopen(filename, "a");

    if (file == NULL){
        printf("Unable to open file %s\n", filename);
        return;
    }

    if (index == 0){
        fprintf(file, "Planet Set:\n\n");
    }

    fprintf(file, "%d.\tName: %s\n", index + 1, planet->name);
    fprintf(file, "\tCircumference: %d\n\n", planet->circumference);

    fclose(file);
}

#endif //ASSIGNMENT_QUESTION_2_TESTINGSTRUCTFILEB_H
```

Below are some explanations of the above code:

In *'Figure 9.1'*, we see the use of the '#include' tag which includes the needed libraries to make the functions work. We also see the use of the '#define' tag which creates a variable as a constant.

```
- Figure 9.1
#include <stdio.h>
#include <string.h>

#define MAX_STRING_LENGTH 64
```

In *'Figure 9.2'*, we see the creation of 2 separate structs, one called Planes, and the other called Planets. These structs act as a data type when stored in the GenSet struct. For each of the 2 structs, functions have been created using the previous declared function pointers.

```
- Figure 9.2
// Structs for Planes
typedef struct Planes {
    char name[MAX_STRING_LENGTH];
    int capacity;
} Planes;

// Structs for Planets
typedef struct Planets {
    char name[MAX_STRING_LENGTH];
    int circumference;
} Planets;

// Returns the size of Planes
size_t getSizeOfPlanes(){
    return sizeof(Planes);
}

// Returns the size of Planets
size_t getSizeOfPlanets(){
    return sizeof(Planets);
}

// Compares Plane sets
int comparePlanes(void *element1, void *element2) {
    Planes *plane1 = (Planes *)element1;
    Planes *plane2 = (Planes *)element2;

    if ((strcmp(plane1->name, plane2->name) == 0) && (plane1->capacity ==
plane2->capacity)) {
        return 1;
    } else {
        return 0;
    }
}

// Compares Planet sets
int comparePlanets(void *element1, void *element2) {
    Planets *planet1 = (Planets *)element1;
    Planets *planet2 = (Planets *)element2;

    if ((strcmp(planet1->name, planet2->name) == 0) && (planet1-
>circumference == planet2->circumference)) {
        return 1;
    } else {
        return 0;
    }
}

// Adds Plane sets to the set
int addPlanesToSet(void *element1, void *element2){
    Planes *plane1 = (Planes *)element1;
    Planes *plane2 = (Planes *)element2;

    strcpy(plane1->name, plane2->name);

    plane1->capacity += plane2->capacity;

    return 1;
}
```

```c
// Adds Planet sets to the set
int addPlanetsToSet(void *element1, void *element2){
    Planets *planet1 = (Planets *)element1;
    Planets *planet2 = (Planets *)element2;

    strcpy(planet1->name, planet2->name);

    planet1->circumference += planet2->circumference;

    return 1;
}

// Displays Plane sets
void displayPlanes(void *element){
    Planes *plane = (Planes *)element;

    printf("Plane name: %s\n", plane->name);
    printf("Plane capacity: %d\n\n", plane->capacity);
}

// Displays Planet sets
void displayPlanets(void *element){
    Planets *planet = (Planets *)element;

    printf("Planet name: %s\n", planet->name);
    printf("Planet circumference: %d\n\n", planet->circumference);
}

// Exports Plane sets
void exportPlanes(void *element, char *filename, int index){
    Planes *plane = (Planes *)element;

    FILE *file = fopen(filename, "a");

    if (file == NULL){
        printf("Unable to open file %s\n", filename);
        return;
    }

    if (index == 0){
        fprintf(file, "Plane Set:\n\n");
    }

    fprintf(file, "%d.\tName: %s\n", index + 1, plane->name);
    fprintf(file, "\tCapacity: %d\n\n", plane->capacity);

    fclose(file);
}

// Exports Planet sets
void exportPlanets(void *element, char *filename, int index){
    Planets *planet = (Planets *)element;

    FILE *file = fopen(filename, "a");

    if (file == NULL){
        printf("Unable to open file %s\n", filename);
        return;
    }

    if (index == 0){
        fprintf(file, "Planet Set:\n\n");
    }

    fprintf(file, "%d.\tName: %s\n", index + 1, planet->name);
    fprintf(file, "\tCircumference: %d\n\n", planet->circumference);

    fclose(file);
}
```

Question 2

Below is the file needed to compile both Task 2A and Task 2B, to run their respective programs.

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.26)
project("Assignment Question 2" C)

set(CMAKE_C_STANDARD 11)

add_executable(Question_2A Question2A.c)
add_executable(Question_2B Question2B.c)

add_library(GenSetLibA SHARED setMethodsA.c) # Creating a shared library
for task A
add_library(GenSetLibB SHARED setMethodsB.c) # Creating a shared library
for task B

target_link_libraries(Question_2A GenSetLibA) # Linking the library to the
executable
target_link_libraries(Question_2B GenSetLibB) # Linking the library to the
executable
```

From the above we can see that a shared library was created for both Task A and Task B to make code easily accessible.

Section 3: How To Run

Question 1:

As stated before, to run on Windows, kindly uncomment lines 6 and 7 in the CMakeLists.txt file, by removing the hashes in front:

```
MATH(EXPR stack_size "16 * 1024 * 1024") # 16 Mb
set(CMAKE_EXE_LINKER_FLAGS "-Wl,--stack,${stack_size}")
```

On other operating systems, the stack size should already be correctly set.

In CLion reload the CMake project, then select 'Question_1' in the configurations tab at the top right and run the program. Here the user can enter their own polynomial, starting root approximations, and number of iterations.

Question 2

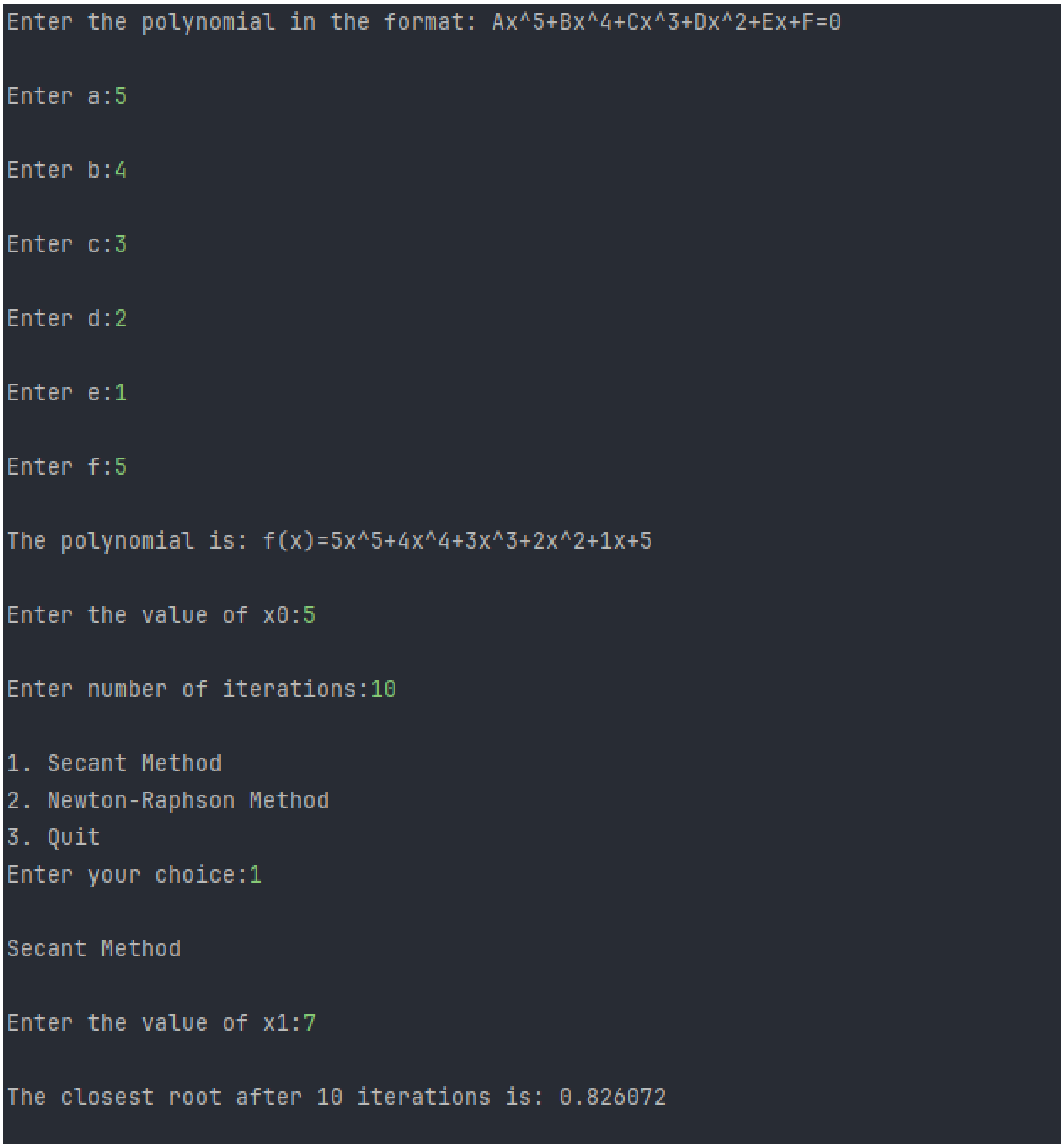
Question 2A
In CLion reload the CMake project, then select 'Question_2A' in the configurations tab at the top right and run the program. In this part the user can choose between entering their own custom integer and string sets or using predefined sets.

Question 2B
In CLion reload the CMake project, then select 'Question_2B' in the configurations tab at the top right and run the program. Here the user is not given an option and predefined sets are used.

Section 4: Test Results

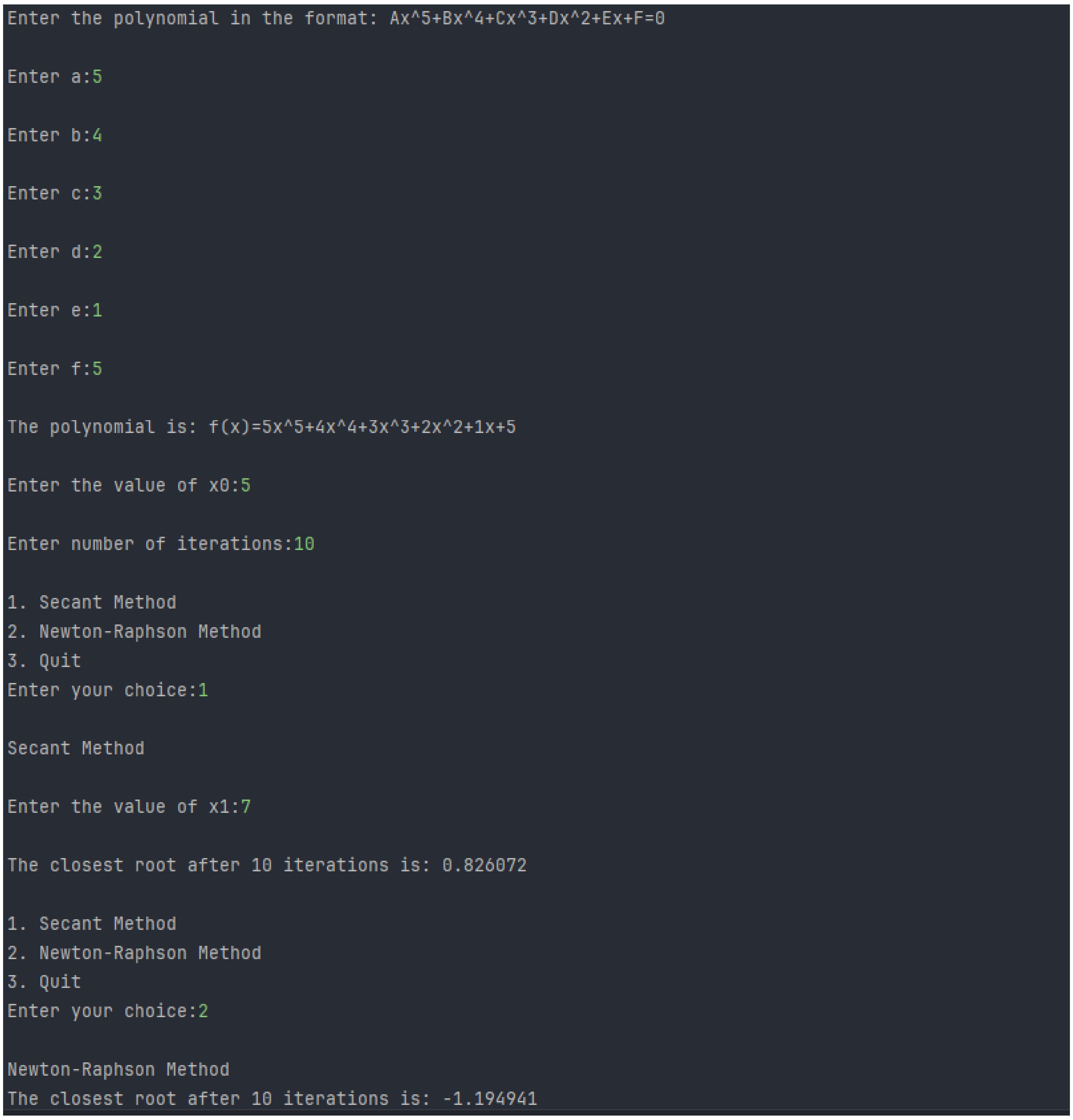
Question 1:

Testing the Secant Method:



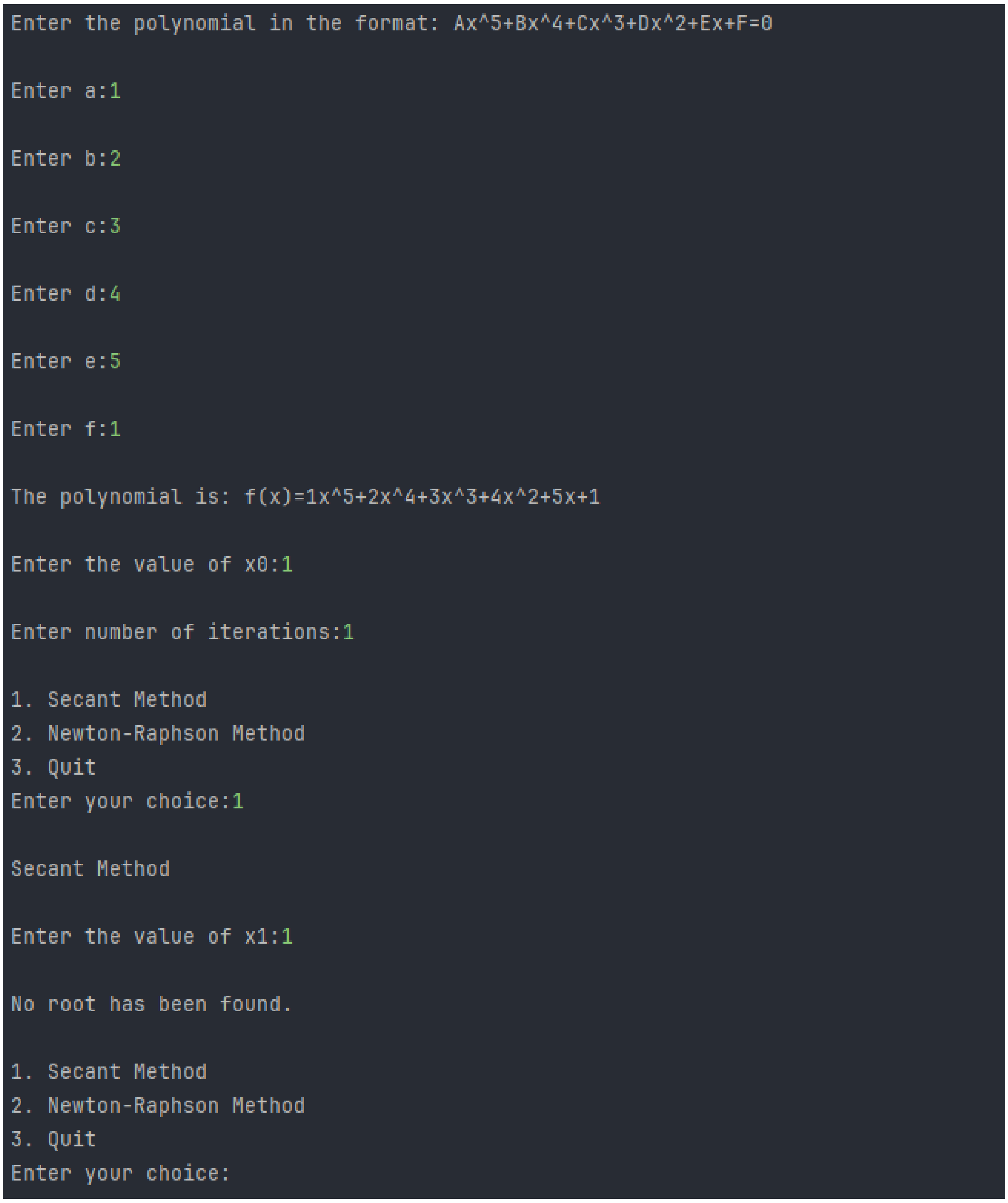
```
Enter the polynomial in the format: Ax^5+Bx^4+Cx^3+Dx^2+Ex+F=0

Enter a:5

Enter b:4

Enter c:3

Enter d:2

Enter e:1

Enter f:5

The polynomial is: f(x)=5x^5+4x^4+3x^3+2x^2+1x+5

Enter the value of x0:5

Enter number of iterations:10

1. Secant Method
2. Newton-Raphson Method
3. Quit
Enter your choice:1

Secant Method

Enter the value of x1:7

The closest root after 10 iterations is: 0.826072
```

Testing the Newton-Raphson Method:



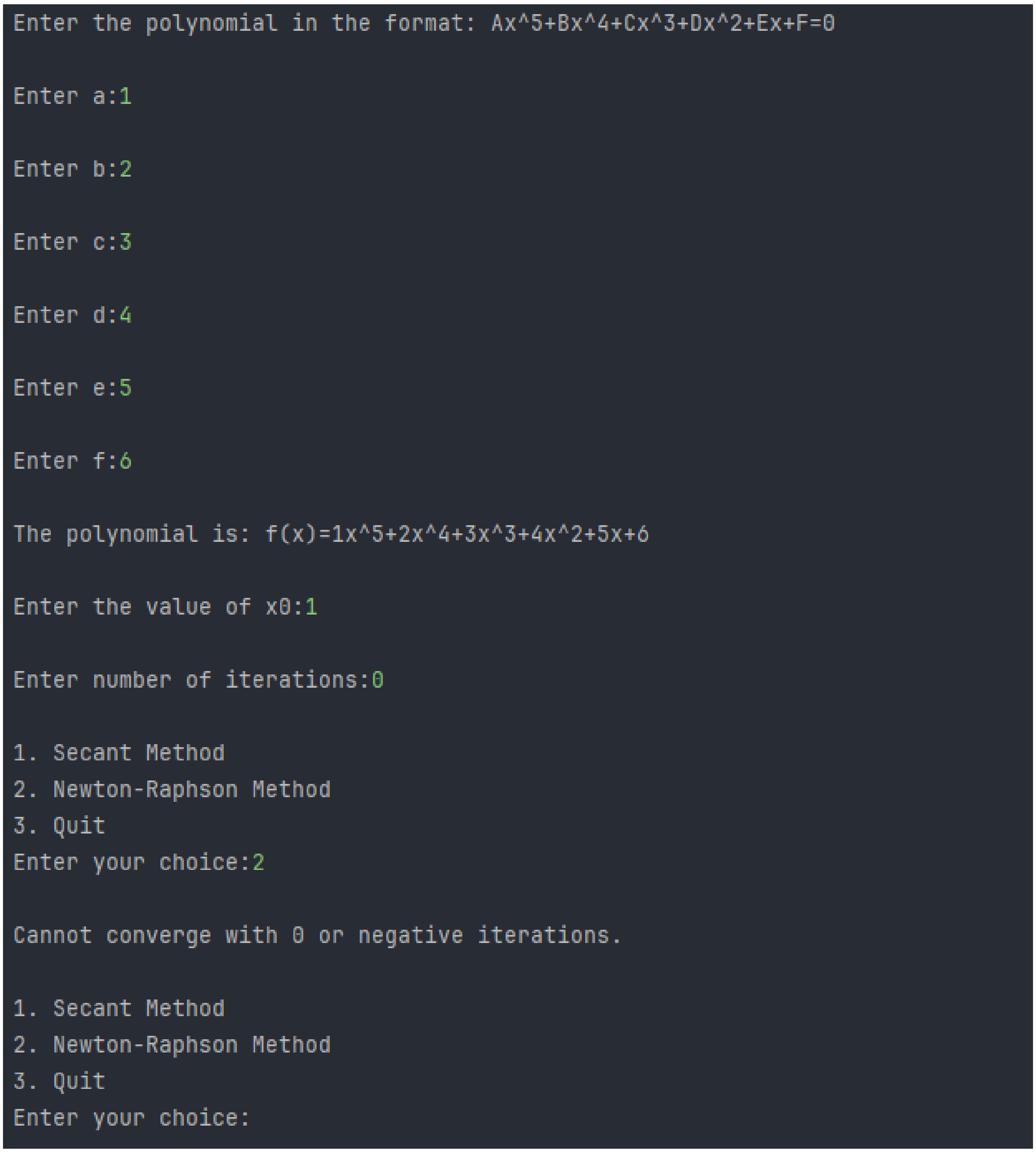
```
Enter the polynomial in the format: Ax^5+Bx^4+Cx^3+Dx^2+Ex+F=0

Enter a:5

Enter b:4

Enter c:3

Enter d:2

Enter e:1

Enter f:5

The polynomial is: f(x)=5x^5+4x^4+3x^3+2x^2+1x+5

Enter the value of x0:5

Enter number of iterations:10

1. Secant Method
2. Newton-Raphson Method
3. Quit
Enter your choice:1

Secant Method

Enter the value of x1:7

The closest root after 10 iterations is: 0.826072

1. Secant Method
2. Newton-Raphson Method
3. Quit
Enter your choice:2

Newton-Raphson Method
The closest root after 10 iterations is: -1.194941
```

Test case of Secant Method with no root:



```
Enter the polynomial in the format: Ax^5+Bx^4+Cx^3+Dx^2+Ex+F=0

Enter a:1

Enter b:2

Enter c:3

Enter d:4

Enter e:5

Enter f:1

The polynomial is: f(x)=1x^5+2x^4+3x^3+4x^2+5x+1

Enter the value of x0:1

Enter number of iterations:1

1. Secant Method
2. Newton-Raphson Method
3. Quit
Enter your choice:1

Secant Method

Enter the value of x1:1

No root has been found.

1. Secant Method
2. Newton-Raphson Method
3. Quit
Enter your choice:
```

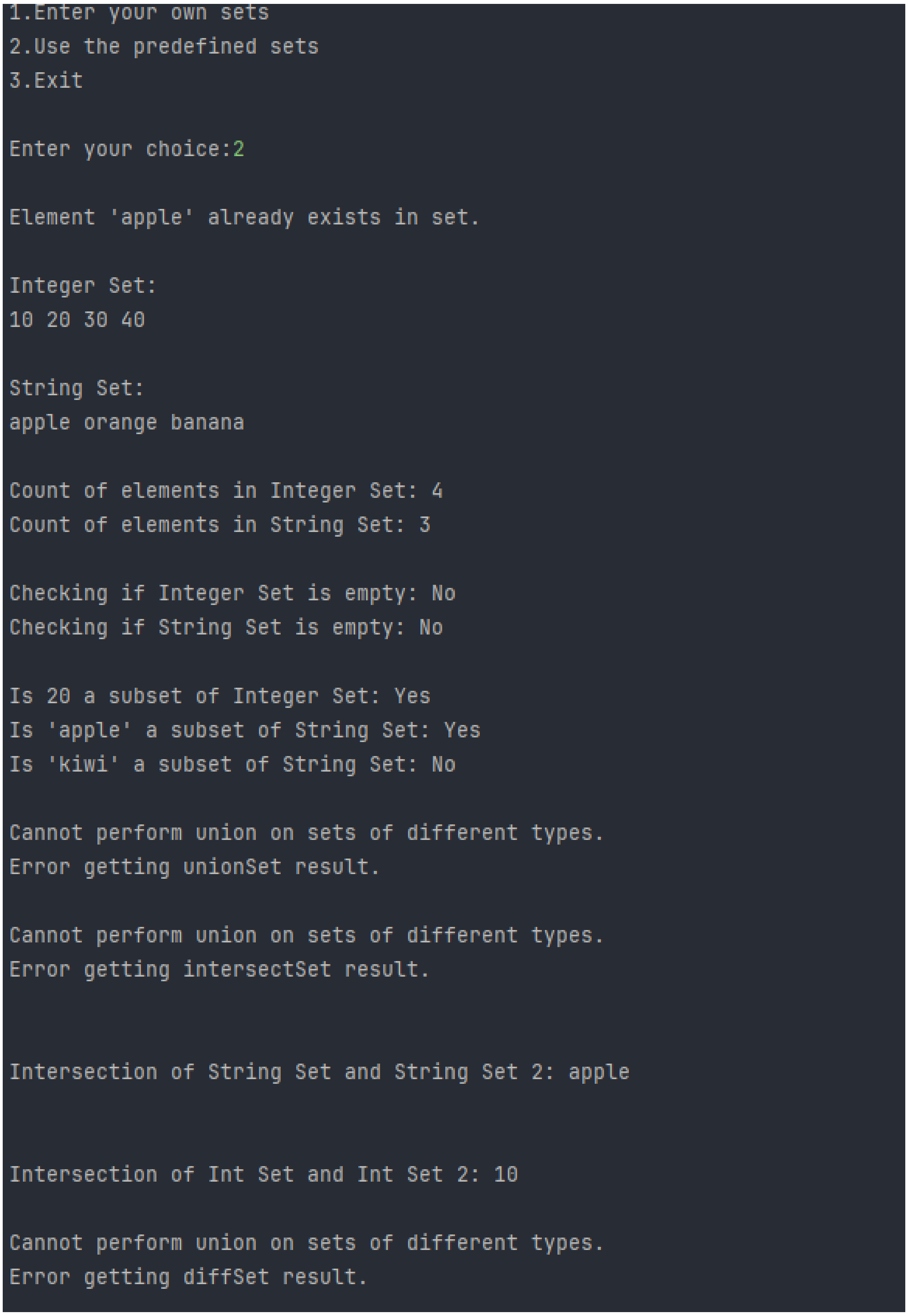
Test case of Newton-Raphson Method with number of iterations not valid:



```
Enter the polynomial in the format: Ax^5+Bx^4+Cx^3+Dx^2+Ex+F=0

Enter a:1

Enter b:2

Enter c:3

Enter d:4

Enter e:5

Enter f:6

The polynomial is: f(x)=1x^5+2x^4+3x^3+4x^2+5x+6

Enter the value of x0:1

Enter number of iterations:0

1. Secant Method
2. Newton-Raphson Method
3. Quit
Enter your choice:2

Cannot converge with 0 or negative iterations.

1. Secant Method
2. Newton-Raphson Method
3. Quit
Enter your choice:
```

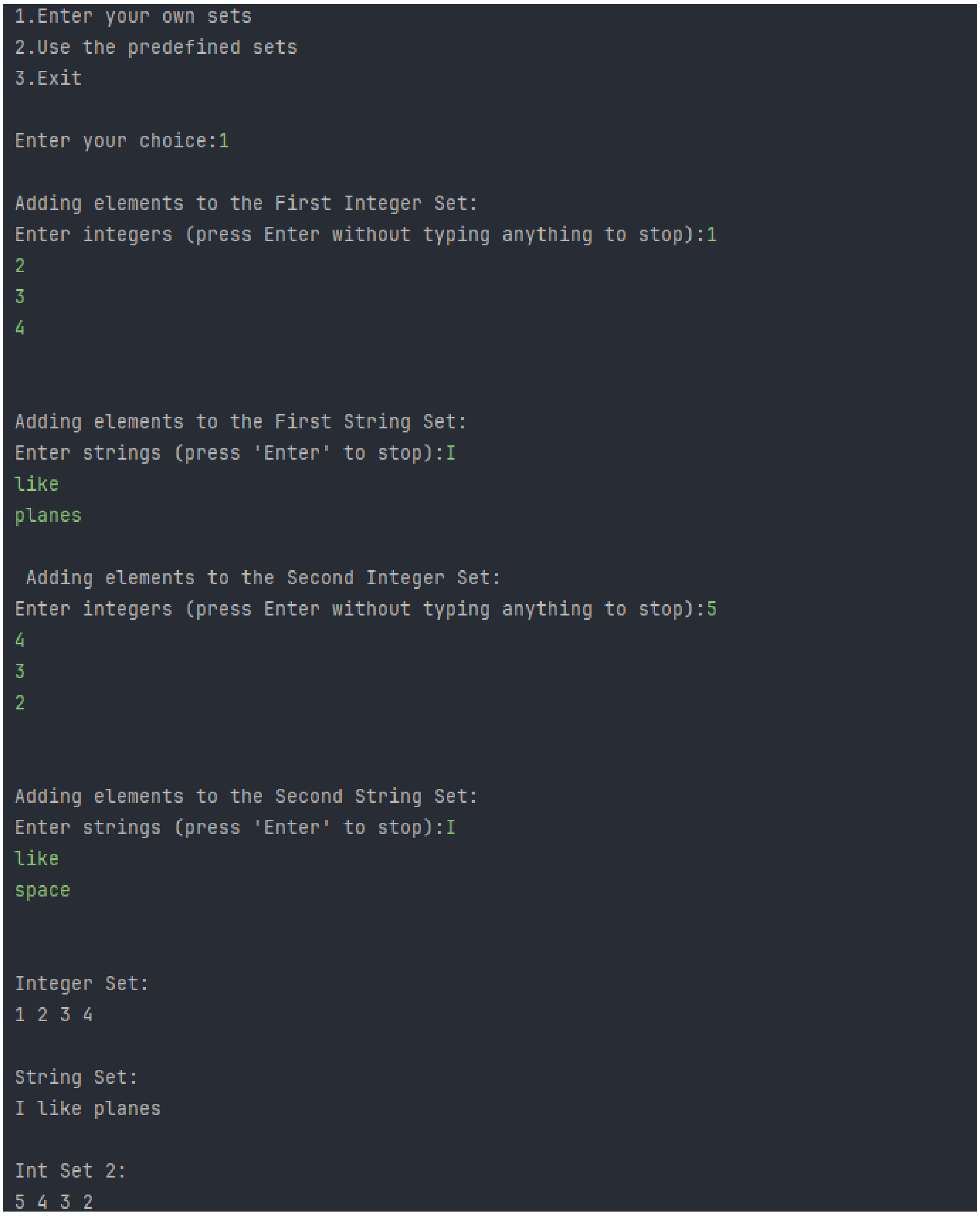
Question 2A:

Predefined sets test case:

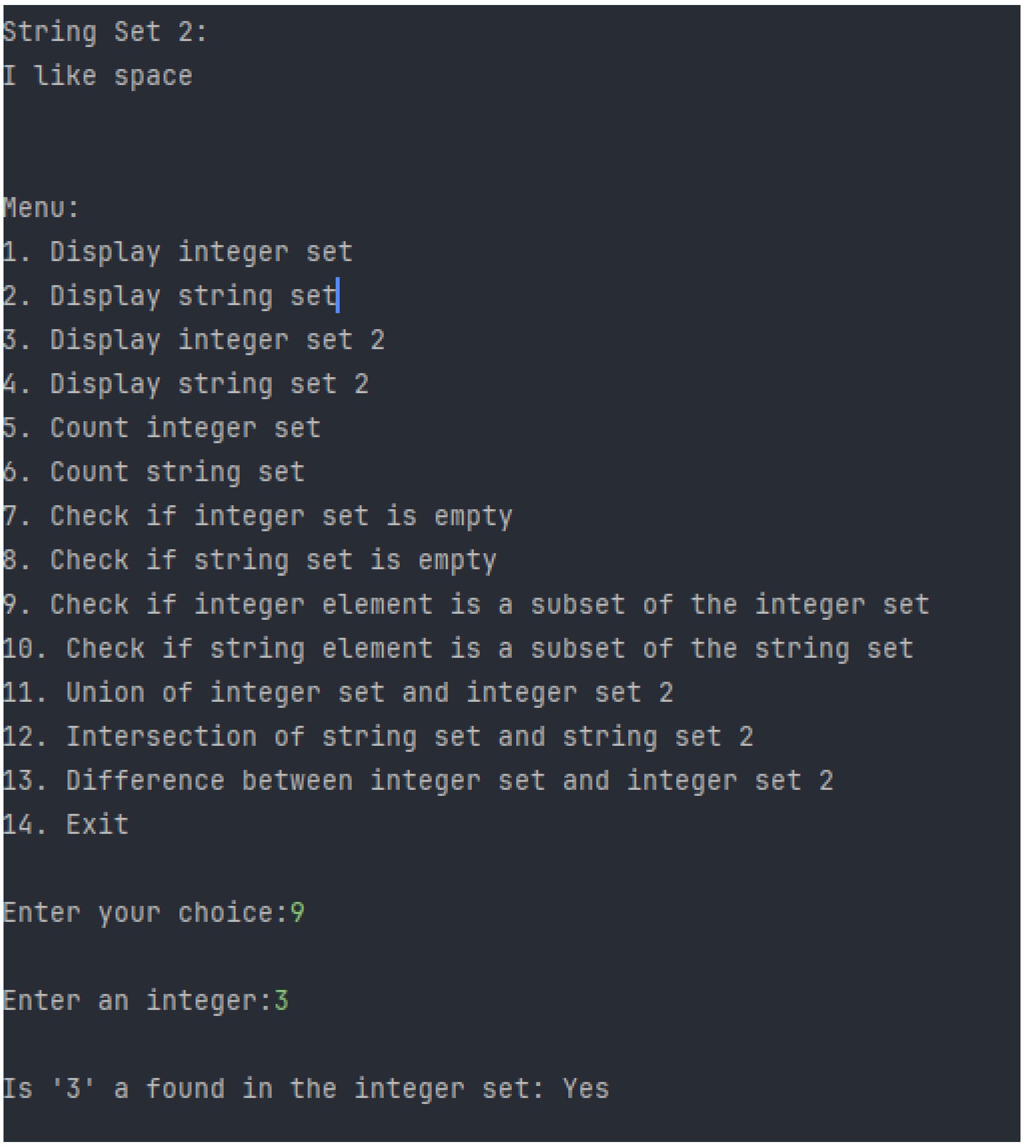


```
1.Enter your own sets
2.Use the predefined sets
3.Exit

Enter your choice:2

Element 'apple' already exists in set.

Integer Set:
10 20 30 40

String Set:
apple orange banana

Count of elements in Integer Set: 4
Count of elements in String Set: 3

Checking if Integer Set is empty: No
Checking if String Set is empty: No

Is 20 a subset of Integer Set: Yes
Is 'apple' a subset of String Set: Yes
Is 'kiwi' a subset of String Set: No

Cannot perform union on sets of different types.
Error getting unionSet result.

Cannot perform union on sets of different types.
Error getting intersectSet result.


Intersection of String Set and String Set 2: apple


Intersection of Int Set and Int Set 2: 10

Cannot perform union on sets of different types.
Error getting diffSet result.
```

User defined sets case:



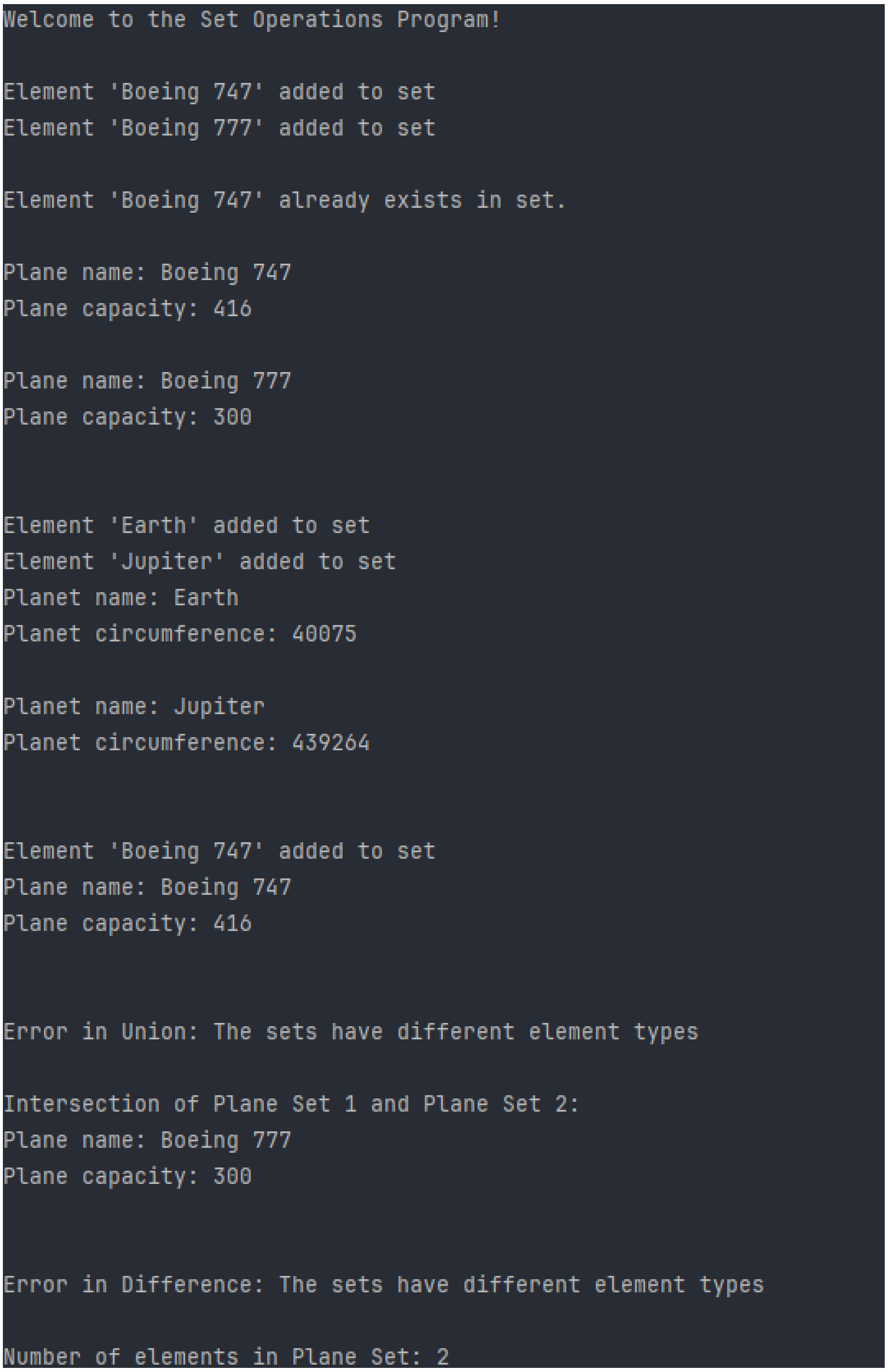
```
1.Enter your own sets
2.Use the predefined sets
3.Exit

Enter your choice:1

Adding elements to the First Integer Set:
Enter integers (press Enter without typing anything to stop):1
2
3
4


Adding elements to the First String Set:
Enter strings (press 'Enter' to stop):I
like
planes

 Adding elements to the Second Integer Set:
Enter integers (press Enter without typing anything to stop):5
4
3
2


Adding elements to the Second String Set:
Enter strings (press 'Enter' to stop):I
like
space


Integer Set:
1 2 3 4

String Set:
I like planes

Int Set 2:
5 4 3 2
```



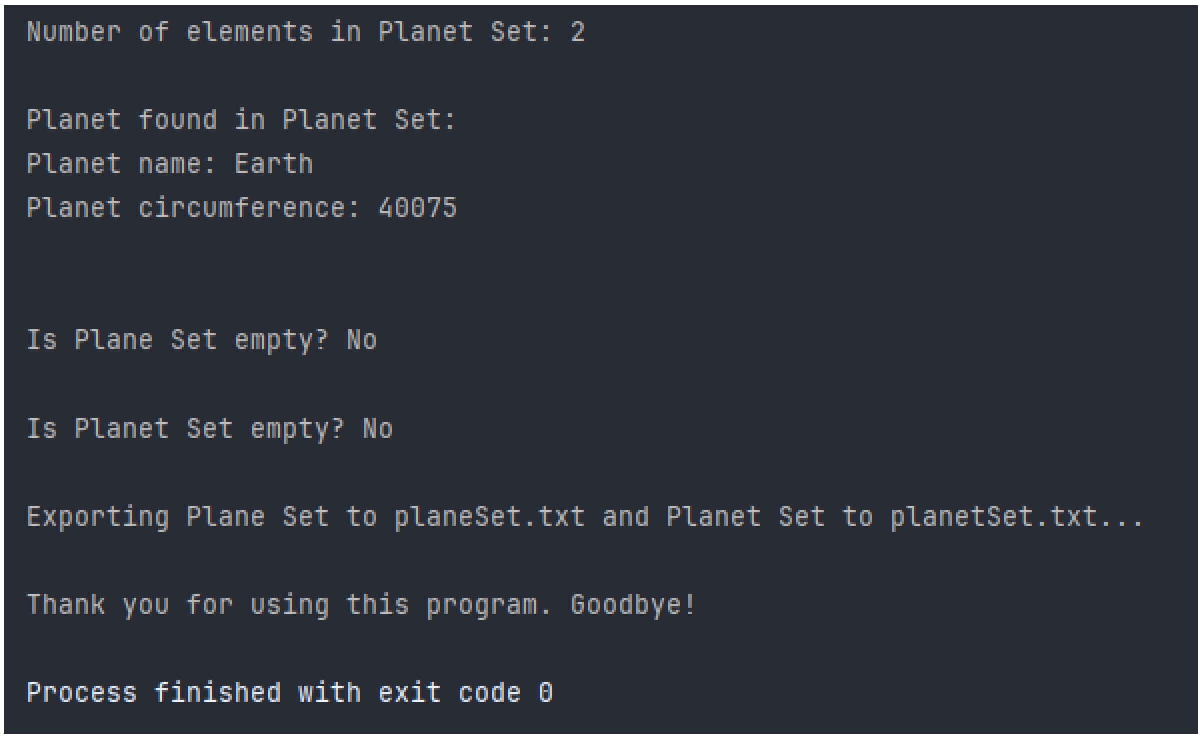
```
String Set 2:
I like space


Menu:
1. Display integer set
2. Display string set
3. Display integer set 2
4. Display string set 2
5. Count integer set
6. Count string set
7. Check if integer set is empty
8. Check if string set is empty
9. Check if integer element is a subset of the integer set
10. Check if string element is a subset of the string set
11. Union of integer set and integer set 2
12. Intersection of string set and string set 2
13. Difference between integer set and integer set 2
14. Exit

Enter your choice:9

Enter an integer:3

Is '3' a found in the integer set: Yes
```

Question 2B:

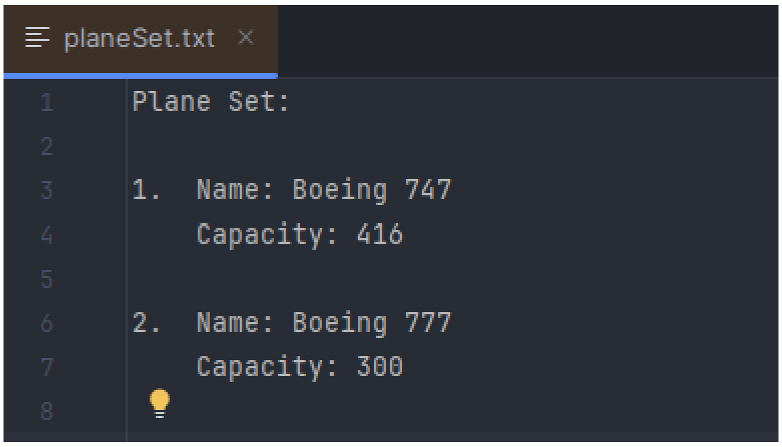
Test case for 2B:



```
Welcome to the Set Operations Program!

Element 'Boeing 747' added to set
Element 'Boeing 777' added to set

Element 'Boeing 747' already exists in set.

Plane name: Boeing 747
Plane capacity: 416

Plane name: Boeing 777
Plane capacity: 300


Element 'Earth' added to set
Element 'Jupiter' added to set
Planet name: Earth
Planet circumference: 40075

Planet name: Jupiter
Planet circumference: 439264


Element 'Boeing 747' added to set
Plane name: Boeing 747
Plane capacity: 416


Error in Union: The sets have different element types

Intersection of Plane Set 1 and Plane Set 2:
Plane name: Boeing 777
Plane capacity: 300


Error in Difference: The sets have different element types

Number of elements in Plane Set: 2
```

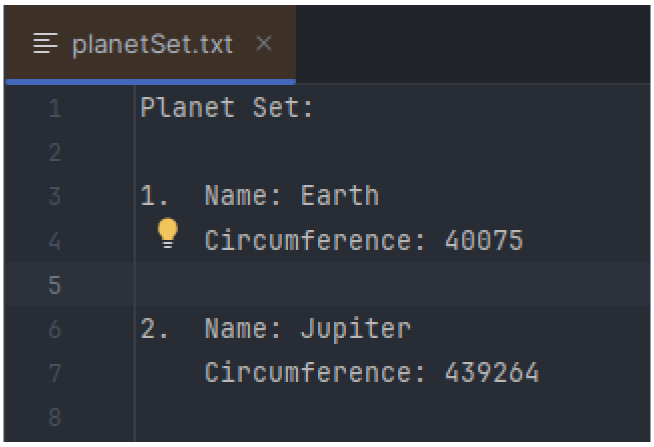


```
Number of elements in Planet Set: 2

Planet found in Planet Set:
Planet name: Earth
Planet circumference: 40075


Is Plane Set empty? No

Is Planet Set empty? No

Exporting Plane Set to planeSet.txt and Planet Set to planetSet.txt...

Thank you for using this program. Goodbye!

Process finished with exit code 0
```



planeSet.txt

```
Plane Set:

1.   Name: Boeing 747
     Capacity: 416

2.   Name: Boeing 777
     Capacity: 300

```



planetSet.txt

```
Planet Set:

1.   Name: Earth
     Circumference: 40075

2.   Name: Jupiter
     Circumference: 439264

```

Section 5: Assumptions and Limitations

Question 1:

In Task 1, we assume the following:

- The polynomial only contains integers when inputting A – F.

Question 2A:

In Task 2A we assume the following:

- The user can only enter integer and string sets.

Question 2B:

In Task 2B we assume the following:

- There is no user input, and the testing is only done using predefined sets.

Section 6: Comments and Conclusion

To conclude, this project, after multiple iterations of the code, was completed successfully to all the user requirements requested in the assignment PDF.

In Question 1, both Secant Method and Newton-Raphson Methods work according to the requirements given. The only limitation in this task being the fact that the user is only able to enter integers into the polynomial, apart from the root approximation.

In Question 2A and 2B the code included the respective functions according to the requirements, with only a few limitations such as in Question 2A the user can only enter integer and string sets, whereas in Question 2B there are no predefined sets.

The inclusion of a shared library was done for both 2A and 2B to ensure that code can be reused, and code can be dynamically allocated.