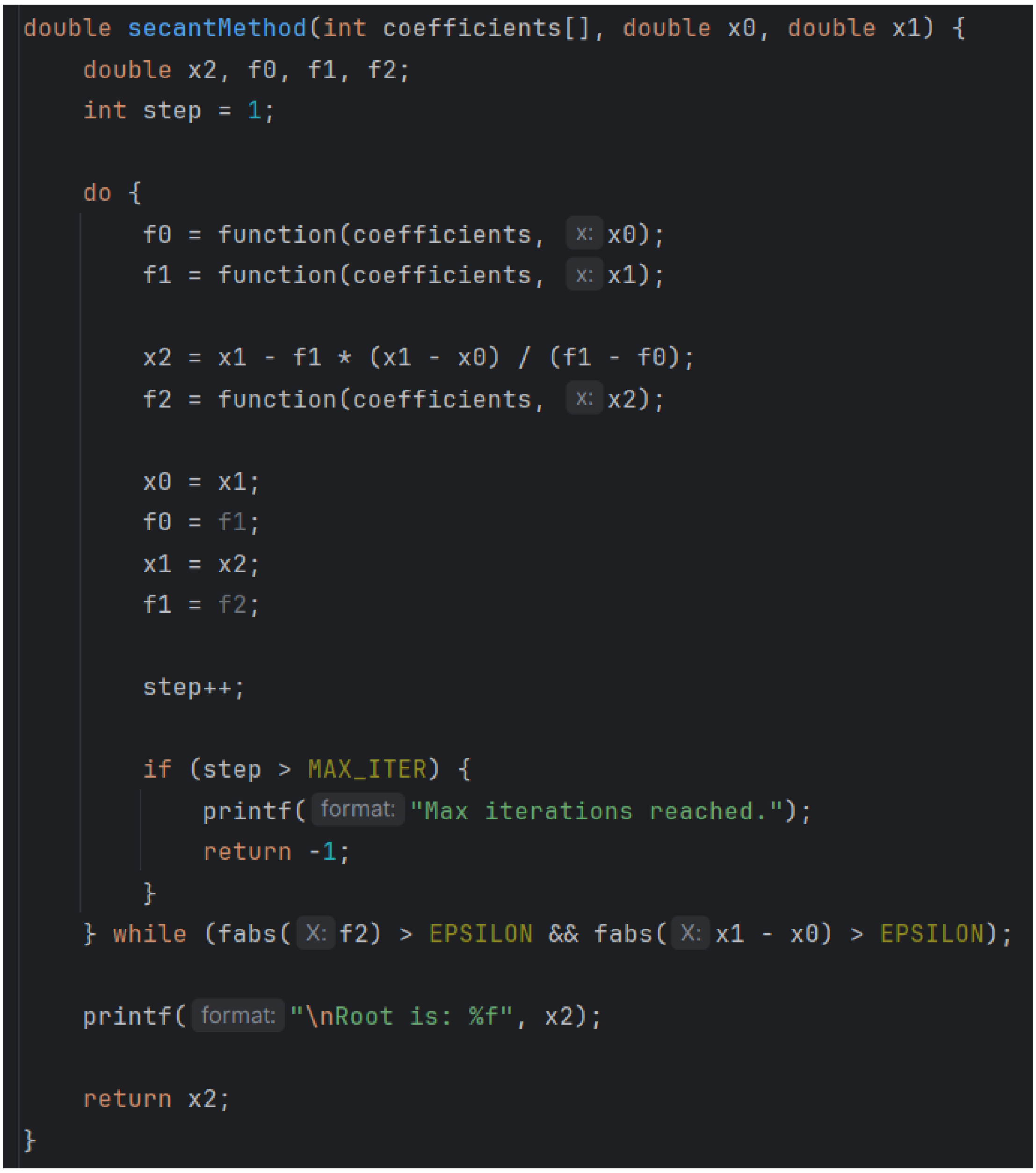
CPS1011 Programming Principles (in C)

Assignment 2023/24

Otmar Nezdařil – 2312649

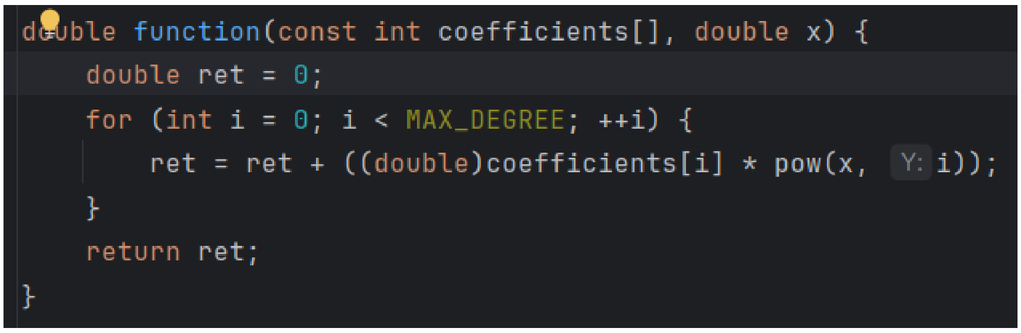
https://gitlab.com/otmarnezdaril/cps1011-assigment

Part 1a



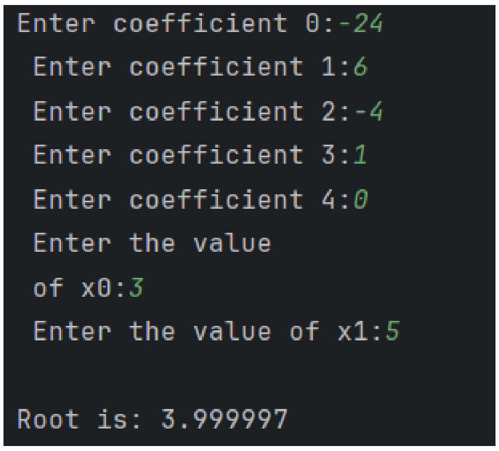
```c
double secantMethod(int coefficients[], double x0, double x1) {
    double x2, f0, f1, f2;
    int step = 1;

    do {
        f0 = function(coefficients, x: x0);
        f1 = function(coefficients, x: x1);

        x2 = x1 - f1 * (x1 - x0) / (f1 - f0);
        f2 = function(coefficients, x: x2);

        x0 = x1;
        f0 = f1;
        x1 = x2;
        f1 = f2;

        step++;

        if (step > MAX_ITER) {
            printf( format: "Max iterations reached.");
            return -1;
        }
    } while (fabs( X: f2) > EPSILON && fabs( X: x1 - x0) > EPSILON);

    printf( format: "\nRoot is: %f", x2);

    return x2;
}
```

Secant function, passing to it coefficients and 0x, x1 initials guesses. For evaluation calling function function(). Using double for more precise calculations (same for evaluating function, derivation and Newton-Raphson method).



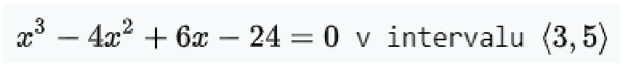
```
double function(const int coefficients[], double x) {
    double ret = 0;
    for (int i = 0; i < MAX_DEGREE; ++i) {
        ret = ret + ((double)coefficients[i] * pow(x,  Y: i));
    }
    return ret;
}
```

Evaluating the function

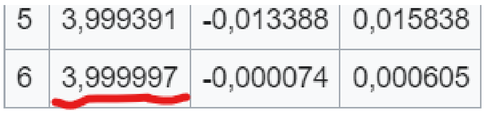


```
Enter coefficient 0:-24
 Enter coefficient 1:6
 Enter coefficient 2:-4
 Enter coefficient 3:1
 Enter coefficient 4:0
 Enter the value
 of x0:3
 Enter the value of x1:5

Root is: 3.999997
```

My output for example from wikipedia.

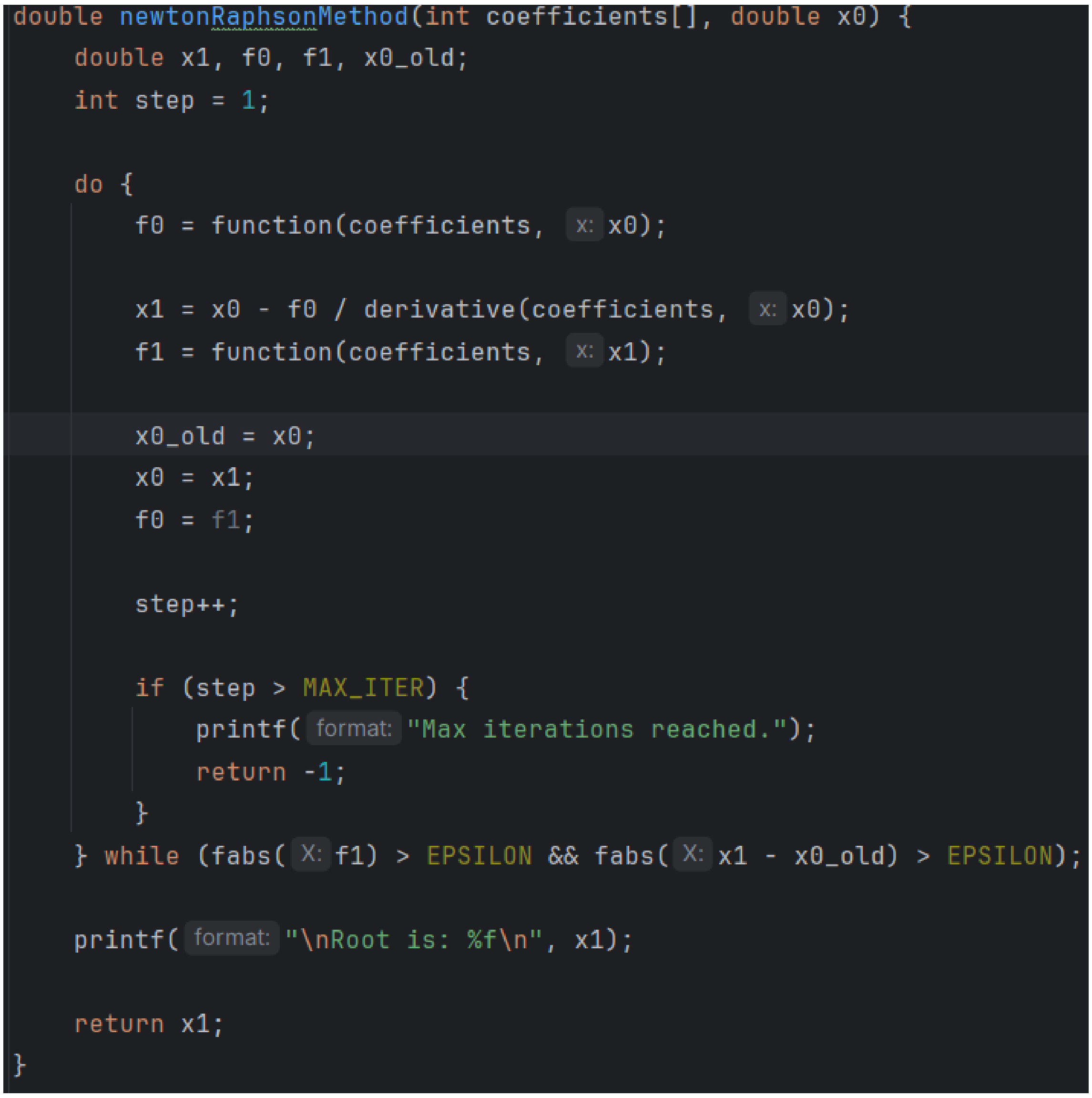


$x^3 - 4x^2 + 6x - 24 = 0$ v intervalu $\langle 3, 5 \rangle$



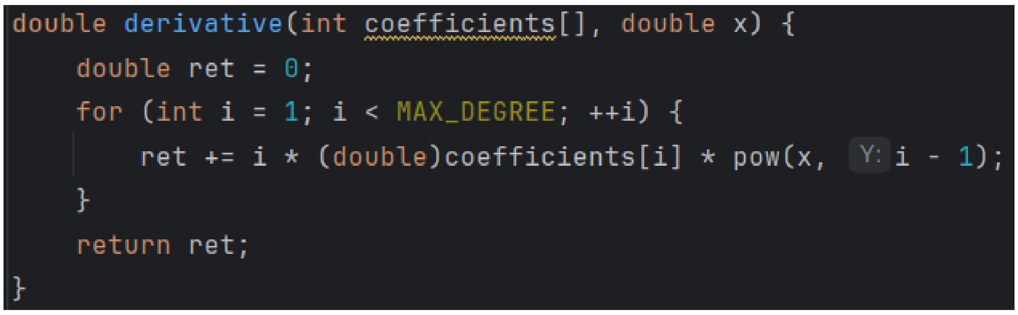
5	3,999391	-0,013388	0,015838
6	3,999997	-0,000074	0,000605

Part 1b



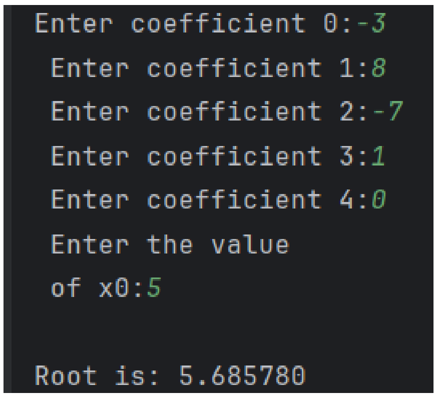
```
double newtonRaphsonMethod(int coefficients[], double x0) {
    double x1, f0, f1, x0_old;
    int step = 1;

    do {
        f0 = function(coefficients, x: x0);

        x1 = x0 - f0 / derivative(coefficients, x: x0);
        f1 = function(coefficients, x: x1);

        x0_old = x0;
        x0 = x1;
        f0 = f1;

        step++;

        if (step > MAX_ITER) {
            printf( format: "Max iterations reached.");
            return -1;
        }
    } while (fabs( X: f1) > EPSILON && fabs( X: x1 - x0_old) > EPSILON);

    printf( format: "\nRoot is: %f\n", x1);

    return x1;
}
```

In Newton-Raphson function storing x0_old is needed, because of the evalution of the while condition (else the absolute value of the x1-x0 would be always 0, so less then epsilon)



```
double derivative(int coefficients[], double x) {
    double ret = 0;
    for (int i = 1; i < MAX_DEGREE; ++i) {
        ret += i * (double)coefficients[i] * pow(x, Y: i - 1);
    }
    return ret;
}
```

Derivation function


```
Enter coefficient 0:-3
 Enter coefficient 1:8
 Enter coefficient 2:-7
 Enter coefficient 3:1
 Enter coefficient 4:0
 Enter the value
 of x0:5

Root is: 5.685780
```

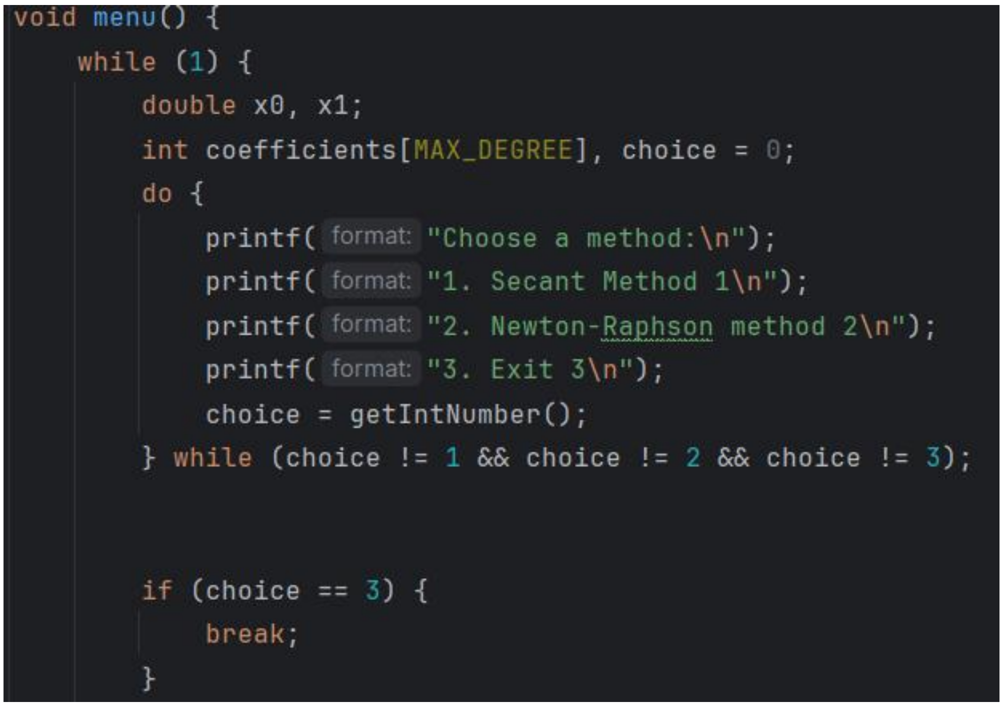
My program output to check if calculations are correct.

1. Use Newton's Method to determine x_2 for $f(x) = x^3 - 7x^2 + 8x - 3$ if $x_0 = 5$

actual root is 5.68577952608963.

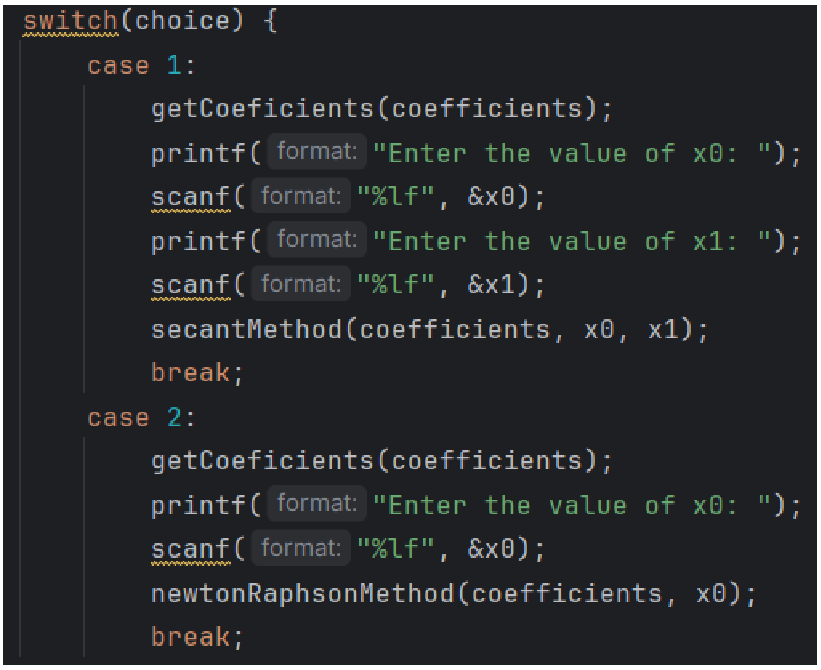
Based on example from the internet, the calculations are correct.

Part 1c



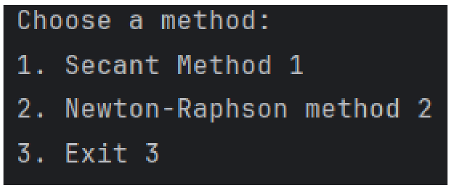
```
void menu() {
    while (1) {
        double x0, x1;
        int coefficients[MAX_DEGREE], choice = 0;
        do {
            printf( format: "Choose a method:\n");
            printf( format: "1. Secant Method 1\n");
            printf( format: "2. Newton-Raphson method 2\n");
            printf( format: "3. Exit 3\n");
            choice = getIntNumber();
        } while (choice != 1 && choice != 2 && choice != 3);


        if (choice == 3) {
            break;
        }
```

Menu is created by in a few print lines and switch. Program repeatedly asks user for the input because of the while(1) and ends by the choice == 3 break condition.



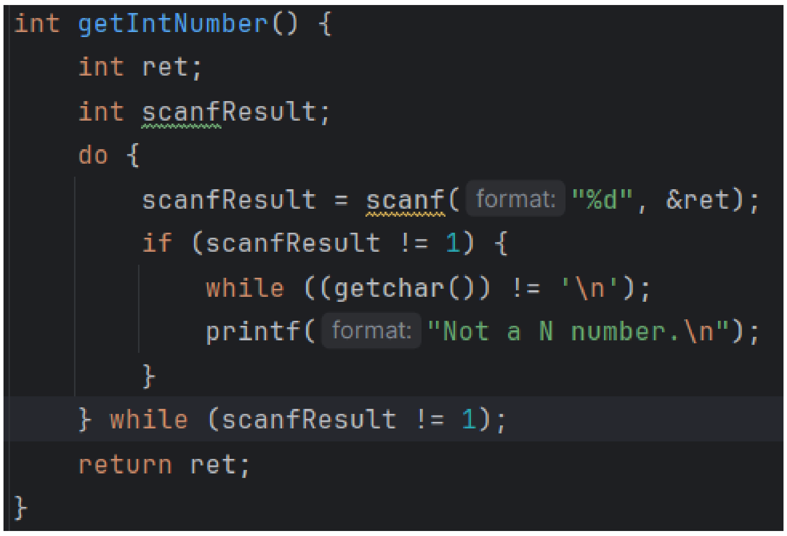
```
switch(choice) {
    case 1:
        getCoeficients(coefficients);
        printf( format: "Enter the value of x0: ");
        scanf( format: "%lf", &x0);
        printf( format: "Enter the value of x1: ");
        scanf( format: "%lf", &x1);
        secantMethod(coefficients, x0, x1);
        break;
    case 2:
        getCoeficients(coefficients);
        printf( format: "Enter the value of x0: ");
        scanf( format: "%lf", &x0);
        newtonRaphsonMethod(coefficients, x0);
        break;
```

Switch for selection option and passing in needed inputs



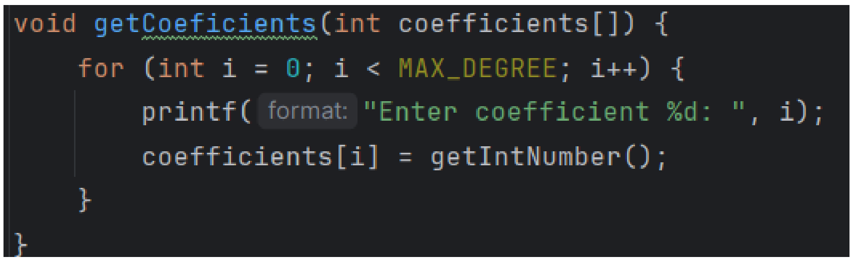
```
Choose a method:
1. Secant Method 1
2. Newton-Raphson method 2
3. Exit 3
```

Snippet of the menu.



```c
int getIntNumber() {
    int ret;
    int scanfResult;
    do {
        scanfResult = scanf( format: "%d", &ret);
        if (scanfResult != 1) {
            while ((getchar()) != '\n');
            printf( format: "Not a N number.\n");
        }
    } while (scanfResult != 1);
    return ret;
}
```

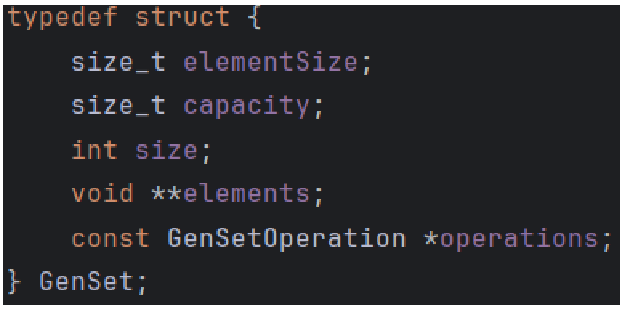
Function to get the valid int from user input, used for menu selection option and getting coefficient for the polynomial function.



```c
void getCoeficients(int coefficients[]) {
    for (int i = 0; i < MAX_DEGREE; i++) {
        printf( format: "Enter coefficient %d: ", i);
        coefficients[i] = getIntNumber();
    }
}
```

Getting all coefficients inputs validated by getInNumber().

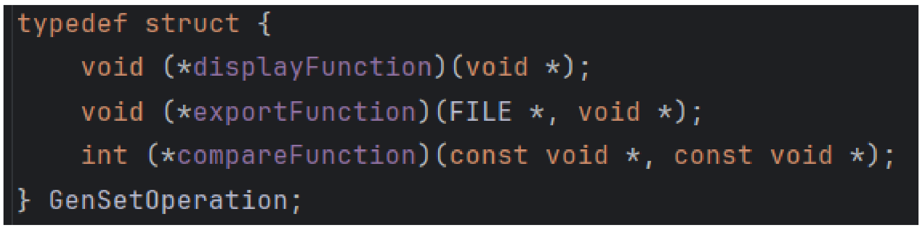
Part 2a



```c
typedef struct {
    size_t elementSize;
    size_t capacity;
    int size;
    void **elements;
    const GenSetOperation *operations;
} GenSet;
```

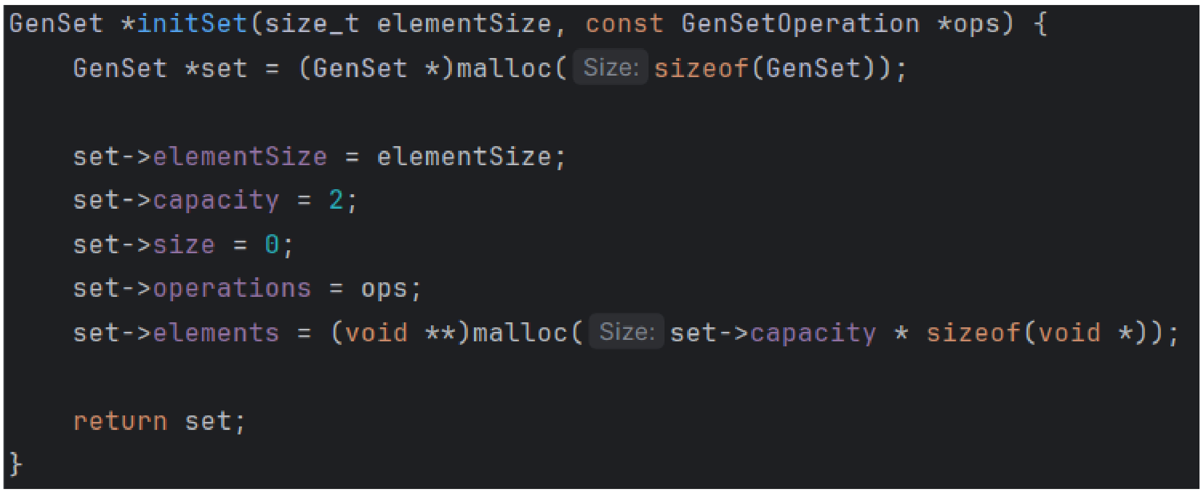
My GenSet consist of:

- Element size: size of each element stored in the set
- Capacity of whole set (maximum elements size in memory)
- Size: counter of elements (useful for knowing elements counts without going throw all of them or the array)
- Pointer to the pointer of the elements
- Operations: set of the operations used for comparison, displaying and exporting



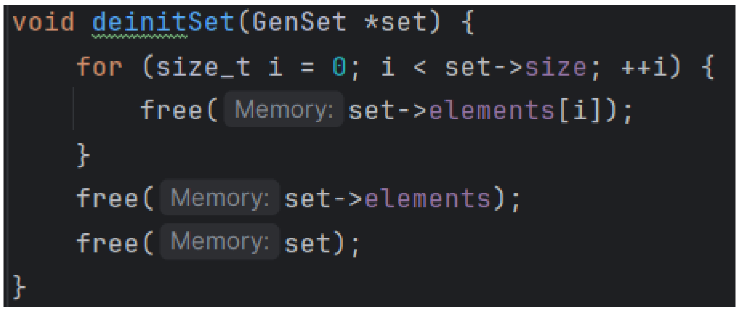
```c
typedef struct {
    void (*displayFunction)(void *);
    void (*exportFunction)(FILE *, void *);
    int (*compareFunction)(const void *, const void *);
} GenSetOperation;
```

Struct to store all to operations needed for the set to be passed at once.



```c
GenSet *initSet(size_t elementSize, const GenSetOperation *ops) {
    GenSet *set = (GenSet *)malloc( Size: sizeof(GenSet));

    set->elementSize = elementSize;
    set->capacity = 2;
    set->size = 0;
    set->operations = ops;
    set->elements = (void **)malloc( Size: set->capacity * sizeof(void *));

    return set;
}
```

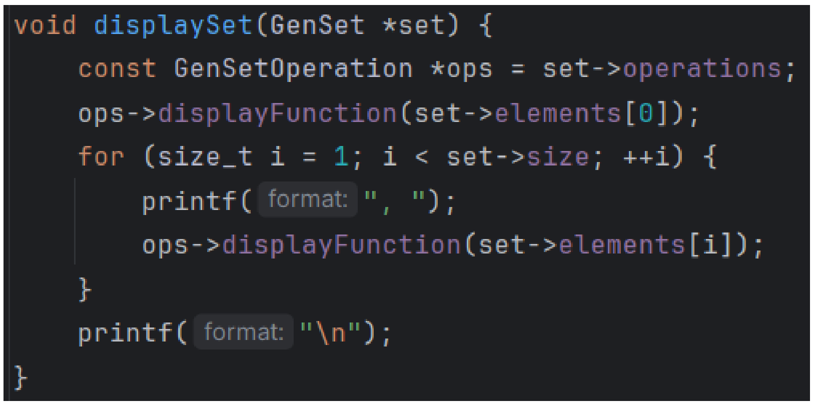
In the initialization, to let the data be stored on a heap, I allocated memory for the struct and for the elements (in the size of 2x elements)



```c
void deinitSet(GenSet *set) {
    for (size_t i = 0; i < set->size; ++i) {
        free( Memory: set->elements[i]);
    }
    free( Memory: set->elements);
    free( Memory: set);
}
```

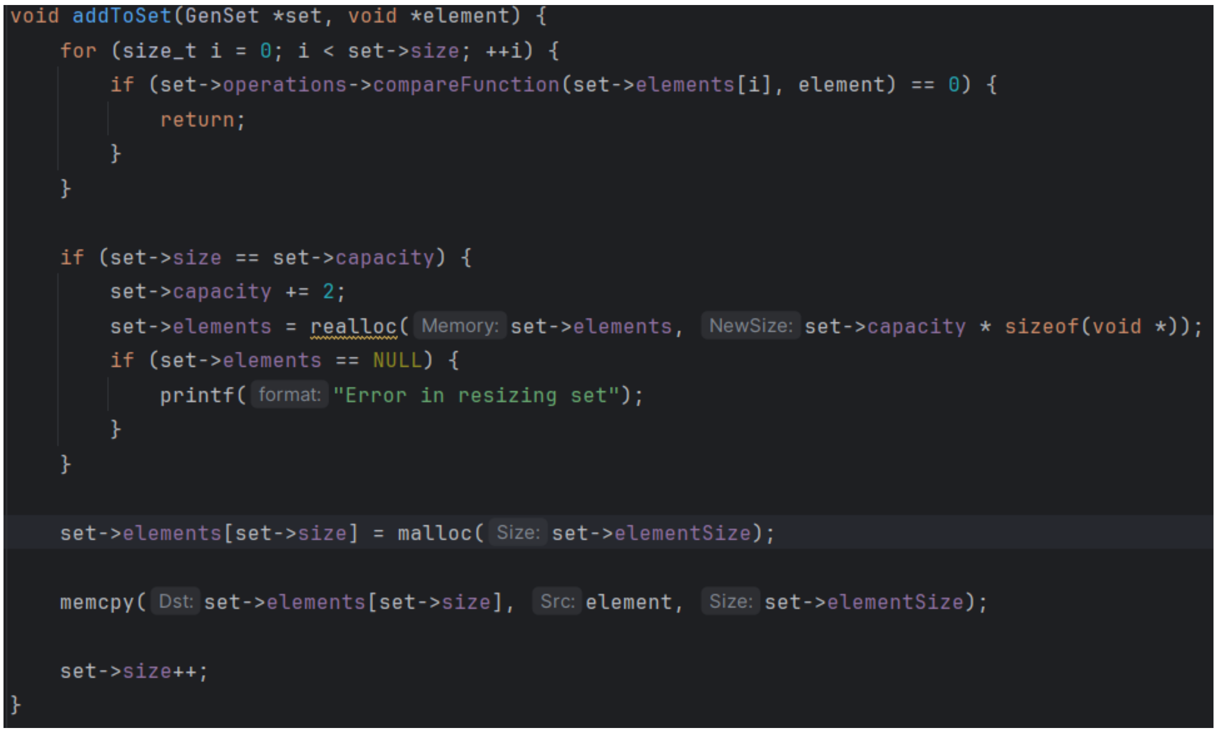
Int deinit, I need to deallocate each element, pointer to them and a whole set since I allocated all of it.

First of all, I need to check if the element already exists in the set, then check if there is free space in the set (eventually extended it if needed – reallocate the memory), then allocate space for the new element and copy the new element in the set.

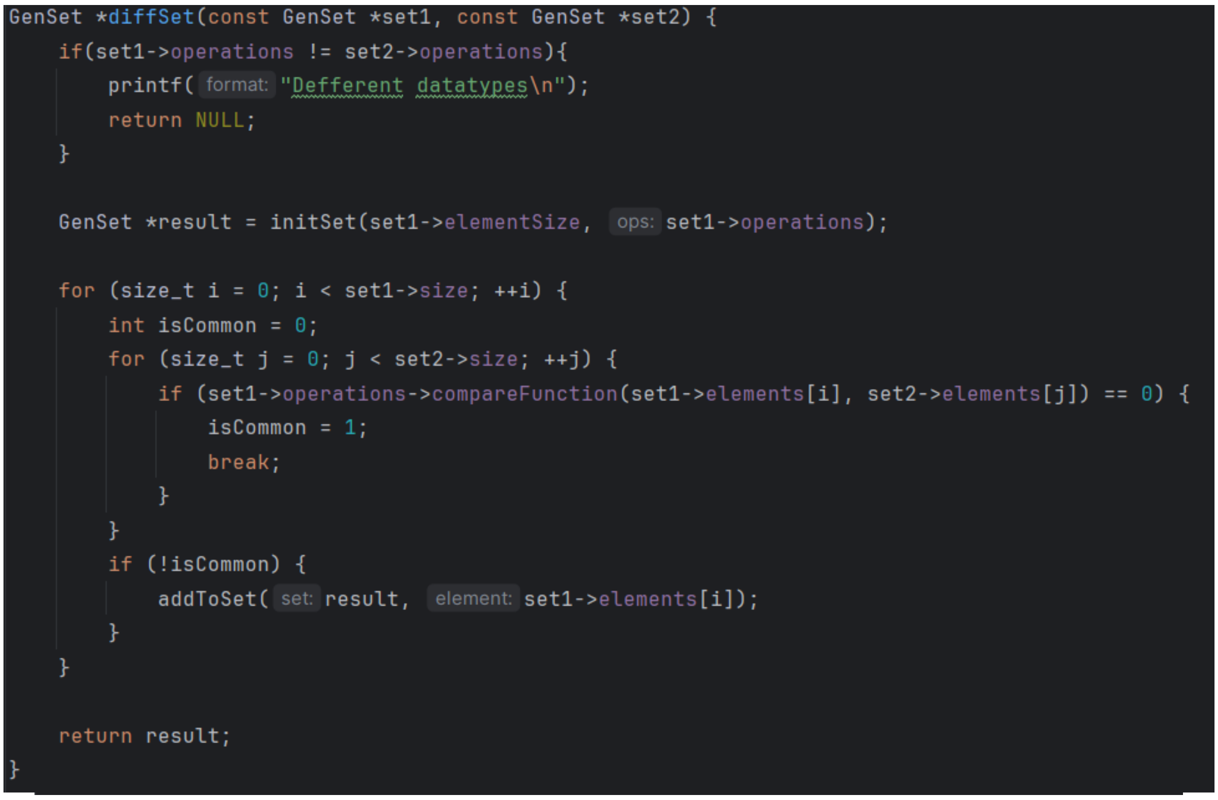


```c
void displaySet(GenSet *set) {
    const GenSetOperation *ops = set->operations;
    ops->displayFunction(set->elements[0]);
    for (size_t i = 1; i < set->size; ++i) {
        printf( format: ", ");
        ops->displayFunction(set->elements[i]);
    }
    printf( format: "\n");
}
```

Displaying set through loop and calling assigned display function to the set.

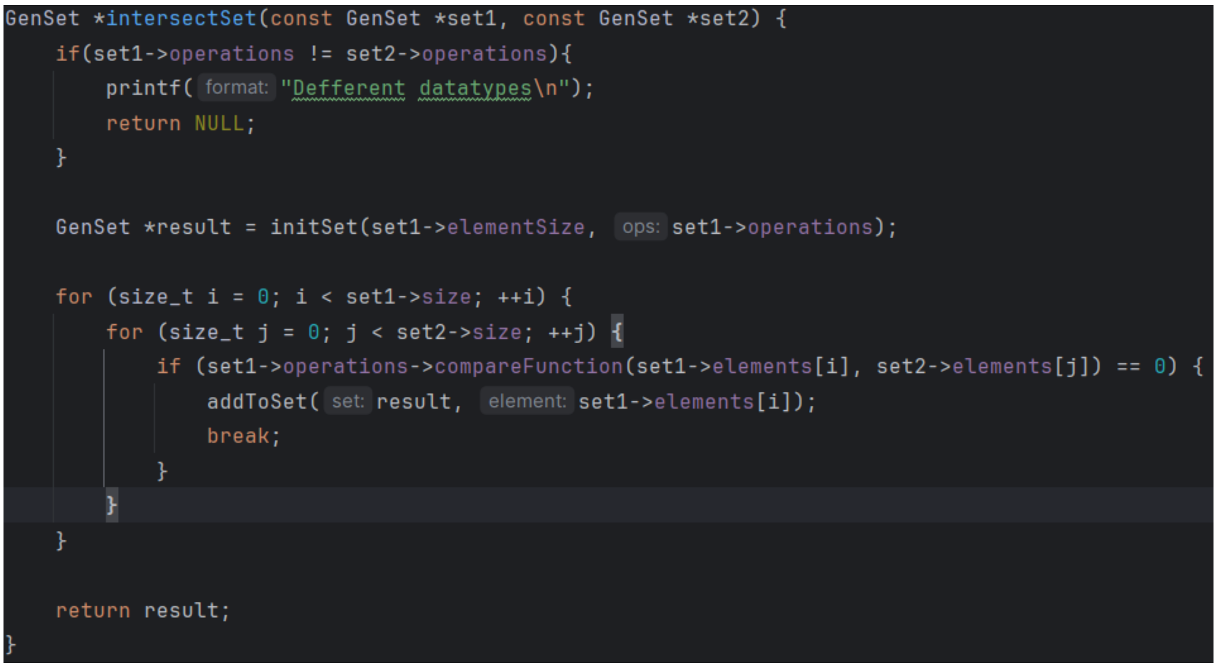


```c
void addToSet(GenSet *set, void *element) {
    for (size_t i = 0; i < set->size; ++i) {
        if (set->operations->compareFunction(set->elements[i], element) == 0) {
            return;
        }
    }

    if (set->size == set->capacity) {
        set->capacity += 2;
        set->elements = realloc( Memory: set->elements,  NewSize: set->capacity * sizeof(void *));
        if (set->elements == NULL) {
            printf( format: "Error in resizing set");
        }
    }

    set->elements[set->size] = malloc( Size: set->elementSize);

    memcpy( Dst: set->elements[set->size],  Src: element,  Size: set->elementSize);

    set->size++;
}
```



```c
GenSet *diffSet(const GenSet *set1, const GenSet *set2) {
    if(set1->operations != set2->operations){
        printf( format: "Defferent datatypes\n");
        return NULL;
    }

    GenSet *result = initSet(set1->elementSize,  ops: set1->operations);

    for (size_t i = 0; i < set1->size; ++i) {
        int isCommon = 0;
        for (size_t j = 0; j < set2->size; ++j) {
            if (set1->operations->compareFunction(set1->elements[i], set2->elements[j]) == 0) {
                isCommon = 1;
                break;
            }
        }
        if (!isCommon) {
            addToSet( set: result,  element: set1->elements[i]);
        }
    }

    return result;
}
```

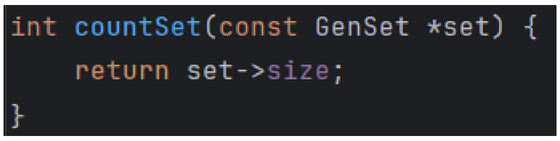
Checking if the same sets are set datatype by comparing operations (predicting the operations assigned to the set are correct, since I directly don't store sets datatype). Adding all elements from both sets to the array. Unique occurrence of each element is guaranteed by the addToSet() function.



```c
GenSet *intersectSet(const GenSet *set1, const GenSet *set2) {
    if(set1->operations != set2->operations){
        printf( format: "Defferent datatypes\n");
        return NULL;
    }

    GenSet *result = initSet(set1->elementSize,  ops: set1->operations);

    for (size_t i = 0; i < set1->size; ++i) {
        for (size_t j = 0; j < set2->size; ++j) {
            if (set1->operations->compareFunction(set1->elements[i], set2->elements[j]) == 0) {
                addToSet( set: result,  element: set1->elements[i]);
                break;
            }
        }
    }

    return result;
}
```

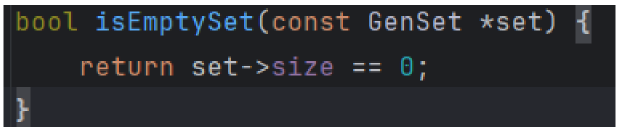
Again, same check from the beginning. Order of the sets passed to the function doesn't matter, just adding each element from one set the returned set if it occurs in the second set.

Kind of reverse function of the intersection, just add it the returned set if it doesn't occurs if the element from one set is not find in the other set.



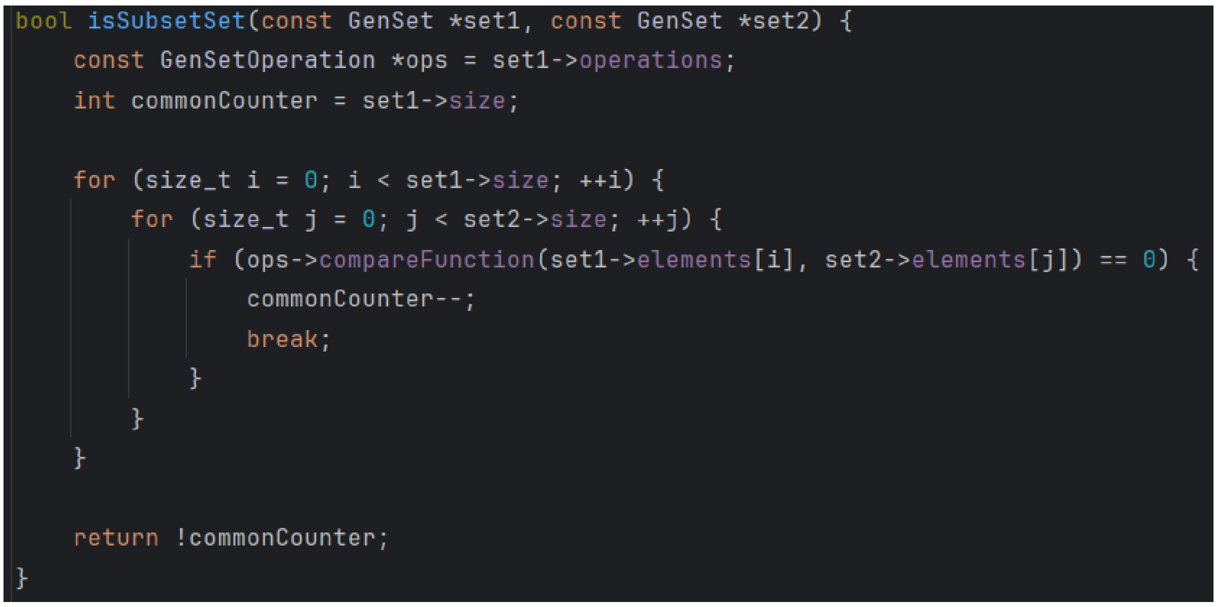
```c
int countSet(const GenSet *set) {
    return set->size;
}
```

Since I store the size counter, this function just returns the size counter.



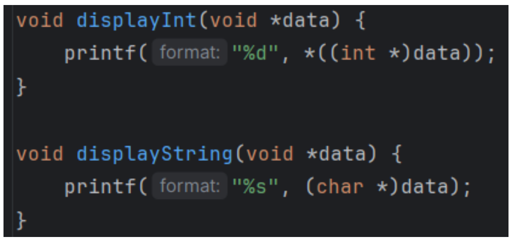
```c
bool isEmptySet(const GenSet *set) {
    return set->size == 0;
}
```

isEmptySet works() works on similar principle, since the program store size

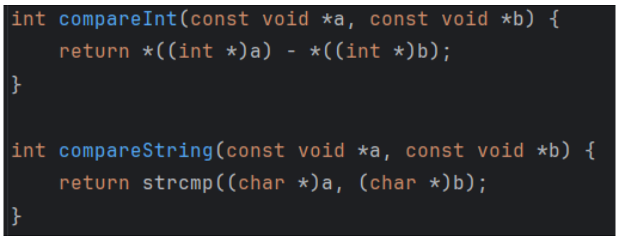


```c
bool isSubsetSet(const GenSet *set1, const GenSet *set2) {
    const GenSetOperation *ops = set1->operations;
    int commonCounter = set1->size;

    for (size_t i = 0; i < set1->size; ++i) {
        for (size_t j = 0; j < set2->size; ++j) {
            if (ops->compareFunction(set1->elements[i], set2->elements[j]) == 0) {
                commonCounter--;
                break;
            }
        }
    }

    return !commonCounter;
}
```

Subset function counts (subtracts from set size) element occurrence in the other set. Function returns bool, so returns "opposite value of the remaining elements" not found in set. (for 0 remaining returned value is true).



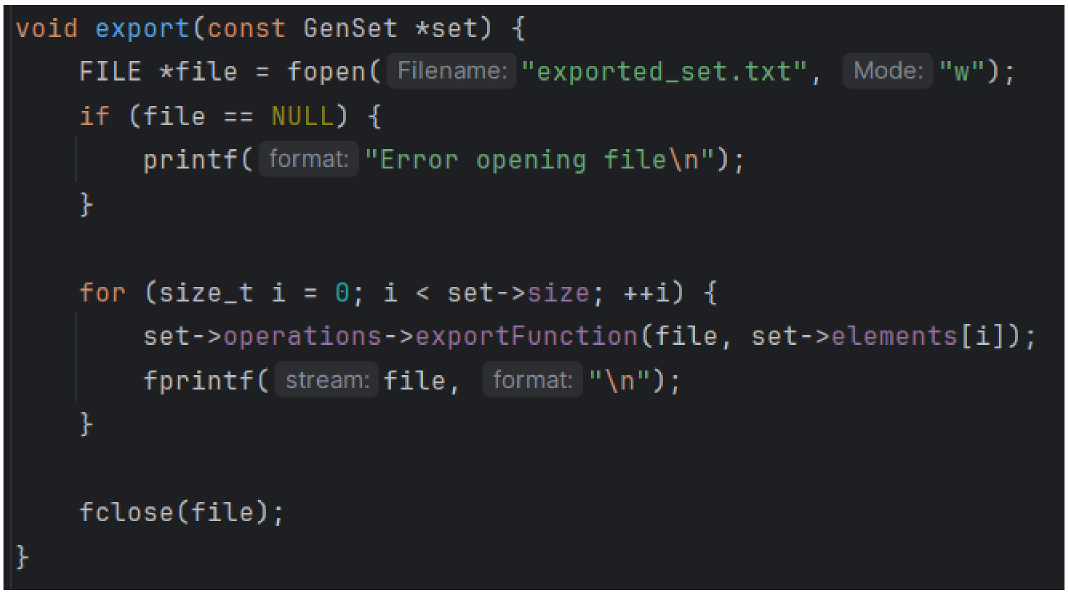
```c
void displayInt(void *data) {
    printf( format: "%d", *((int *)data));
}

void displayString(void *data) {
    printf( format: "%s", (char *)data);
}
```



```c
int compareInt(const void *a, const void *b) {
    return *((int *)a) - *((int *)b);
}

int compareString(const void *a, const void *b) {
    return strcmp((char *)a, (char *)b);
}
```

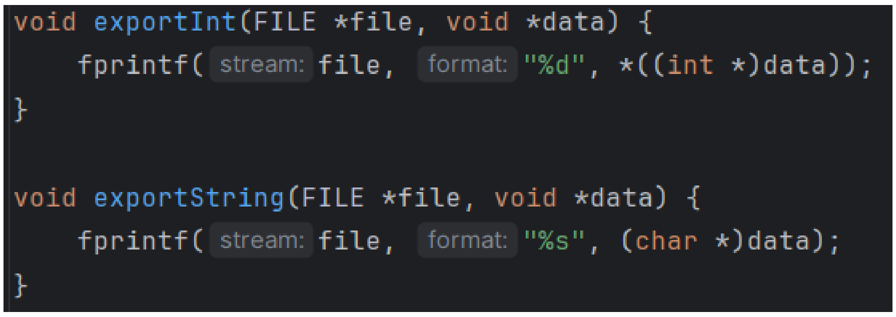
Display function to display implemented data type and compare function used for intersection, diff, etc.

Part 2b



```c
void export(const GenSet *set) {
    FILE *file = fopen( Filename: "exported_set.txt",  Mode: "w");
    if (file == NULL) {
        printf( format: "Error opening file\n");
    }

    for (size_t i = 0; i < set->size; ++i) {
        set->operations->exportFunction(file, set->elements[i]);
        fprintf( stream: file,  format: "\n");
    }

    fclose(file);
}
```

Export function opens file and for each element calls export function. End by proper closing the file.



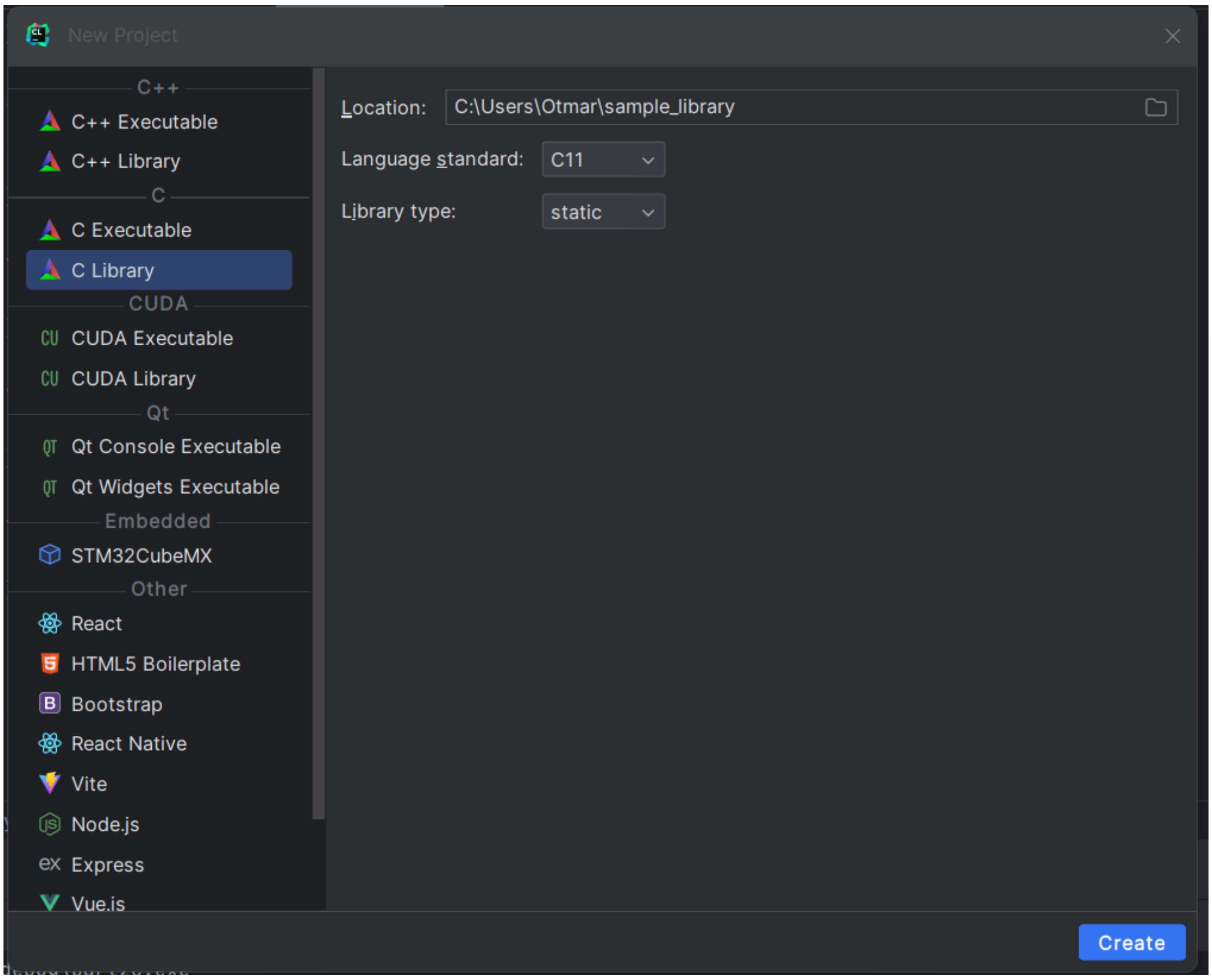
```c
void exportInt(FILE *file, void *data) {
    fprintf( stream: file,  format: "%d", *((int *)data));
}

void exportString(FILE *file, void *data) {
    fprintf( stream: file,  format: "%s", (char *)data);
}
```

Examples of export functions. Input parameters are file and data.

Part 2c

Abstract Data Type interface is implemented even in the part 2a, since this is first solution that I found out how to works with different data types (if I wont just use the if or switch in each function that works differently with different data types).

Shared library I have created and compiled by creating new project in CLion with library.

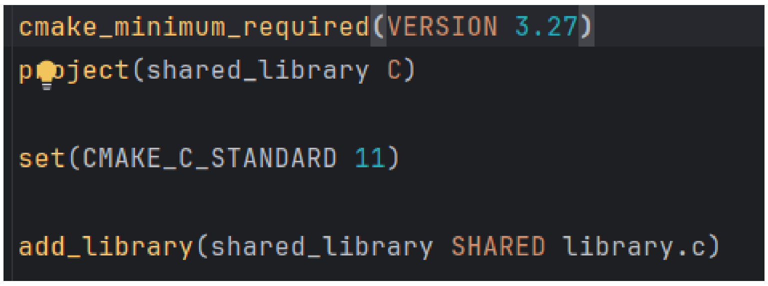


Location: C:\Users\Otmar\sample_library

Language standard: C11

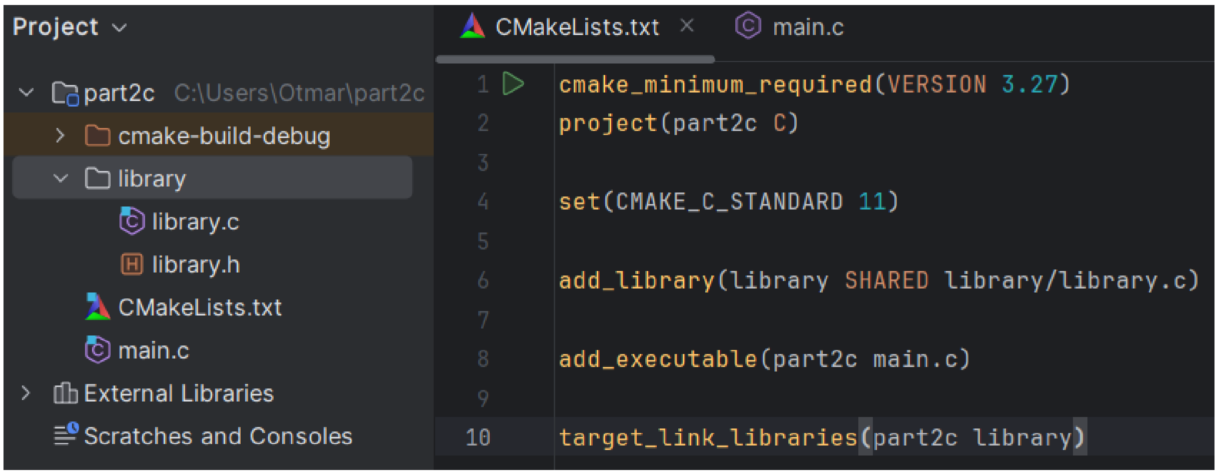
Library type: static

Since CMakeList code is already here



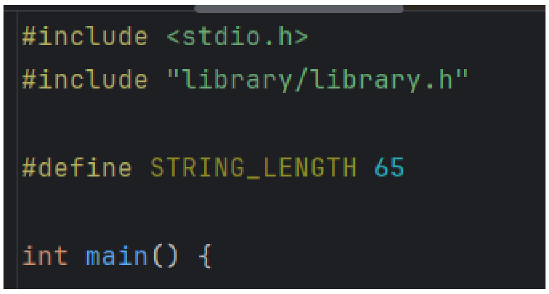
```
cmake_minimum_required(VERSION 3.27)
project(shared_library C)

set(CMAKE_C_STANDARD 11)

add_library(shared_library SHARED library.c)
```

I have, just copied and modified this code and put it in my project (together with the library file and header file)



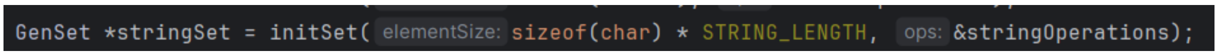
```
cmake_minimum_required(VERSION 3.27)
project(part2c C)

set(CMAKE_C_STANDARD 11)

add_library(library SHARED library/library.c)

add_executable(part2c main.c)

target_link_libraries(part2c library)
```

CMakeList.txt code



```
#include <stdio.h>
#include "library/library.h"

#define STRING_LENGTH 65

int main() {
```

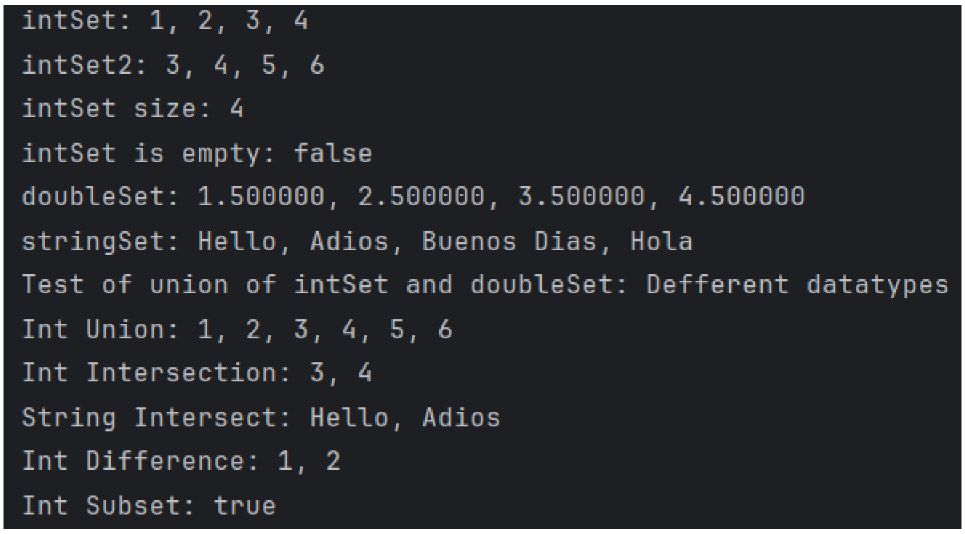
Start of my main.c (including just the header file in the code)



```
GenSet *stringSet = initSet( elementSize: sizeof(char) * STRING_LENGTH,  ops: &stringOperations);
```

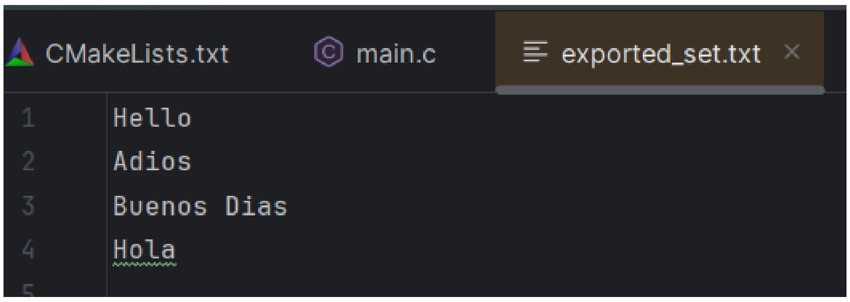
STRING_LENGHT used for the fixed 64 char strings from the task 2b.

Tests



```
intSet: 1, 2, 3, 4
intSet2: 3, 4, 5, 6
intSet size: 4
intSet is empty: false
doubleSet: 1.500000, 2.500000, 3.500000, 4.500000
stringSet: Hello, Adios, Buenos Dias, Hola
Test of union of intSet and doubleSet: Defferent datatypes
Int Union: 1, 2, 3, 4, 5, 6
Int Intersection: 3, 4
String Intersect: Hello, Adios
Int Difference: 1, 2
Int Subset: true
```

Testing all the operations. To test, if all datatypes work fine, I have implemented operations for double as well.



```
Hello
Adios
Buenos Dias
Hola
```

Export function outputs