Лабораторна робота №9. Виключення

Тема: Виключення.

Мета: Навчитись розробляти програми з реалізацією виключень

ВИМОГИ

1.1 Інформація про розробника:

- Кліщов Б. Р.
- КІТ 102.8а

1.2 Загальне завдання

У файлі розміщена інформація про N масивів. В першому рядку міститься інформація про кількість масивів, у кожній наступній – інформація про кількість елементів в кожному масиві та власне дані масиву. Необхідно реалізувати програму, що виконує перераховані нижче дії, причому кожна з них, в окремій функції, поки користувач не введе замість назви файлу рядок \exit Дії, що має виконувати програма такі:

- введення з клавіатури назви вхідного файлу з даними;

- читання даних з файлу;

- виконання індивідуального завдання;

- введення з клавіатури імені вихідного файлу;

- запис результату операції у файл;

- доступ до елемента за індексом слід винести в окрему функцію, що виконує перевірку на можливість виходу за межі масиву.

Слід окремо звернути увагу, що при обробці виключення цикл не повинен перериватись.

1.3 Додаткові умови виконання завдання:

- продемонструвати відсутність витоків пам’яті;

- продемонструвати роботу розроблених методів за допомогою модульних тестів (проемулювати роботу користувача с декількома файлами, командою \exit);

- не використовувати конструкцію «using namespace std;», замість цього слід роботи «using» кожного необхідного класу:using std::string, using std::cout

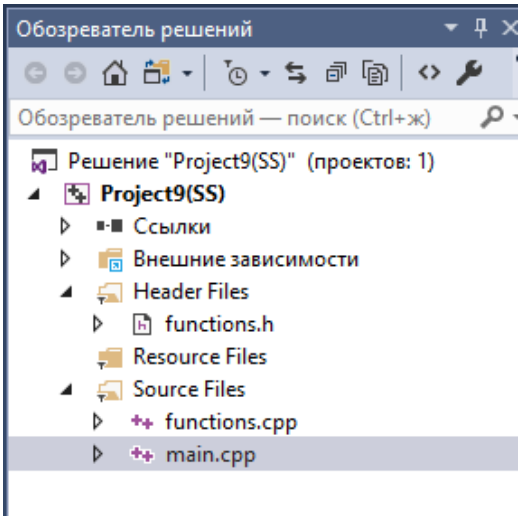
2. ОПИС ПРОГРАМИ

2.1 Функціональне призначення

Програма призначена щоб считувати двовимірний масив з файлу та вазначати суму двовомірних масивів.

2.2 Опис логічної структури

На рисунку № 2 зображена структура програми:



Малюнок №2: Структура програми

2.3 Важливі фрагменти програми

Код програми:

Main():

```
#include"functions.h"

int main()
{
	int **file_data = nullptr;
	int **file_data1 = nullptr;
	int **addition_result = nullptr;
	int rows_in_file;
	int *columns_in_file = nullptr;
	int rows_in_file1;
	int *columns_in_file1 = nullptr;
	int **result;
	int exception;
	string message;
	string file_name;
	do {
```

```
		try
		{
			cout << "Input name of the first file: ";
			cin >> file_name;
			exception = validate_file_name(file_name);
			if (exception == 1)
				throw message = "Invalid name";
			read_file(file_name, &file_data, &rows_in_file, &columns_in_file);

			cout << "Input name of the second file: ";
			cin >> file_name;
			exception = validate_file_name(file_name);
			if (exception == 1)
				throw message = "Invalid name";
			read_file(file_name, &file_data1, &rows_in_file1, &columns_in_file1);

			cout << "The result of addition is:" << endl;
			add_arrays(&file_data, &file_data1, &result, &rows_in_file,
&columns_in_file, &rows_in_file1, &columns_in_file1);

			cout << "Input name of the file to write to: ";
			cin >> file_name;
			exception = validate_file_name(file_name);
			if (exception == 1)
				throw message = "Invalid name";
			write_file(file_name, &result, &rows_in_file, &columns_in_file,
&rows_in_file1, &columns_in_file1);
		}
		catch (...)
		{
			cout << message << endl;
		}
	} while (true);


	return 0;
	 }
```

Functions():

```
#include"functions.h"

int validate_file_name(string file_name)
{
	string message;
	const char *temp_file_name = file_name.c_str();

	if (file_name == "\\exit") {
		_CrtSetReportMode(_CRT_WARN, _CRTDBG_MODE_FILE);
		_CrtSetReportFile(_CRT_WARN, _CRTDBG_FILE_STDERR);
		_CrtSetReportMode(_CRT_ERROR, _CRTDBG_MODE_FILE);
		_CrtSetReportFile(_CRT_ERROR, _CRTDBG_FILE_STDERR);
		_CrtSetReportMode(_CRT_ASSERT, _CRTDBG_MODE_FILE);
		_CrtSetReportFile(_CRT_ASSERT, _CRTDBG_FILE_STDERR);
		_CrtDumpMemoryLeaks();
		exit(0);
	}
	else if (_access(temp_file_name, 0) == -1) {
		return 1;
	}

	return 0;
```

```
}

void read_file(string file_name, int ***file_data, int *rows_in_file, int
**columns_in_file)
{
	try {
		int rows_number;
		int columns_number;

		ifstream file(file_name);

		file >> rows_number;

		*rows_in_file = rows_number;
		*columns_in_file = new int[rows_number]();
		*file_data = new int *[rows_number]();

		for (int i = 0; i < rows_number; i++)
		{
			file >> columns_number;

			(*columns_in_file)[i] = columns_number;
			(*file_data)[i] = new int[columns_number]();
			for (int j = 0; j < columns_number; j++)
			{
				file >> (*file_data)[i][j];
				cout << (*file_data)[i][j] << "\t";
			}
			cout << endl;
		}

		file.close();
	}
	catch (...) {
		cout << "Oh shit, here we go again..." << endl;
	}
}

void add_arrays(int ***file_data, int ***file_data1, int ***result, int *rows_in_file,
int **columns_in_file, int *rows_in_file1, int **columns_in_file1)
{
	int biggest_row;
	int biggest_column;

	if (*rows_in_file >= *rows_in_file1)
	{
		biggest_row = *rows_in_file;
	}
	else
	{
		biggest_row = *rows_in_file1;
	}

	*result = new int *[biggest_row]();

	for (int i = 0; i < biggest_row; i++)
	{
		if ((*columns_in_file)[i] >= (*columns_in_file1)[i])
		{
			(*result)[i] = new int[(*columns_in_file)[i]]();
		}
		else
		{
			(*result)[i] = new int[(*columns_in_file1)[i]]();
		}
```

```
	}

	for (int i = 0; i < *rows_in_file; i++)
	{
		for (int j = 0; j < (*columns_in_file)[i]; j++)
		{
			(*result)[i][j] = (*file_data)[i][j];
		}
	}

	for (int i = 0; i < *rows_in_file1; i++)
	{
		for (int j = 0; j < (*columns_in_file1)[i]; j++)
		{
			(*result)[i][j] += (*file_data1)[i][j];
		}
	}

	for (int i = 0; i < biggest_row; i++)
	{
		if ((*columns_in_file)[i] >= (*columns_in_file1)[i])
		{
			biggest_column = (*columns_in_file)[i];
		}
		else
		{
			biggest_column = (*columns_in_file1)[i];
		}

		for (int j = 0; j < biggest_column; j++)
		{
			cout << (*result)[i][j] << "\t";
		}
		cout << endl;
	}
}

void write_file(string file_name, int ***result, int *rows_in_file, int
**columns_in_file, int *rows_in_file1, int **columns_in_file1)
{
	int biggest_row;
	int biggest_column;
	ofstream file(file_name);

	if (*rows_in_file >= *rows_in_file1)
	{
		biggest_row = *rows_in_file;
	}
	else
	{
		biggest_row = *rows_in_file1;
	}

	for (int i = 0; i < biggest_row; i++)
	{
		if ((*columns_in_file)[i] >= (*columns_in_file1)[i])
		{
			biggest_column = (*columns_in_file)[i];
		}
		else
		{
			biggest_column = (*columns_in_file1)[i];
		}

		for (int j = 0; j < biggest_column; j++)
```

```
            {
                file << (*result)[i][j] << "\t";
            }
            file << endl;
        }

        file.close();
}
```

3 ВАРІАНТИ ВИКОРИСТАННЯ

3.1 Результат роботи функцій

На рисунку № 3 зображено результат роботи програми

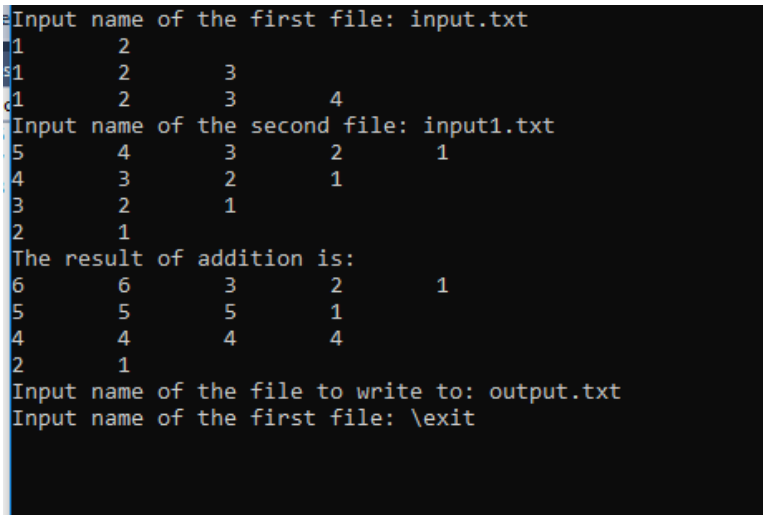


Рисунок № 3. Результат роботи програми

Висновок: Навчився розробляти програми з реалізацією виключень.