НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім ІГОРЯ СІКОРСЬКОГО»

«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»

КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

**КУРСОВА РОБОТА**

з дисципліни Програмування

на тему: «Побудова графіку функції за наданим користувачем рядком,

що містить припустимий опис функції (синтаксичний аналізатор)»

Студентки *1* курсу групи *КА-71*

Спеціальність *124 Системний аналіз*.

*Павлюк Віра*

Керівник старший викладач *Назарчук І.В.*

Національна оцінка

Кількість балів: Оцінка: ECTS

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ (підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

(підпис) (вчене звання, науковий ступінь, прізвище та ініціал

Київ – 2018рік

**ЗМІСТ**

ВСТУП…………………………………………………………………………….….4

РОЗДІЛ 1 Постановка задачі……………………………………………………..5

* 1. Огляд існуючих підходів до розв’язання поставленої задачі……….5
  2. Уточнена постановка задачі на розробку

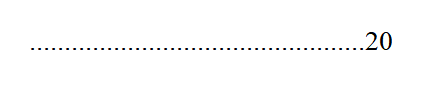
програмного забезпечення....................................................................................5

РОЗДІЛ 2 Розробка програмного продукту………………………………...…...7

* 1. Метод розв’язку задачі……………………………………………..…..7
  2. Алгоритм розв’язку задачі………………………………………….….8

РОЗДІЛ 3 Опис розробленого програмного продукту…………………...…....11

3.1 Опис головних структур і змінних програми……………………..…11

* 1. Опис головних функцій програми……………………………...……13
  2. Опис інтерфейсу……………………………………………………....15
  3. Результати роботи програмного продукту

ВИСНОВКИ………………………………………

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…..………………………………...…..21

Додаток . Текст програми………………………………………………………...22

**ВСТУП**

Дана курсова робота передбачає побудову графіка довільної функції, що містить припустимі операції («+», «-», «\*», «/», «%», «x^(…)», «sin(...)», «cos(…)», «tan(...)», «ctan(…)»).

Метою написання курсової роботи було розробити та реалізувати закінчений програмний продукт для використання під час навчального процесу, а також покращити та закріпити набуті під час вивчення дисципліни «Програмування» теоретичні знання й практичні вміння.

На основі аналізу літературних джерел був реалізований синтаксичний аналізатор, що опрацьовує наданий користувачем рядок та повертає математичний вираз. Була створена функція для обчислення значення ординати для конкретного аргументу. Також реалізована функція для побудови графіку, що приймає заздалегідь підраховані значення координат. Складено структурний алгоритм роботи майбутньої програми та реалізовано його у вигляді програмного продукту, що складається з трьох модулів, з використанням процедурного програмування.

Побудова графіку є допоміжним засобом для розв‘язання багатьох математичних і фізичних задач. Графік наглядно демонструє умову, що полегшує її сприйняття.

При виконанні роботи було використане таке програмне забезпечення: стандартне середовище розробки **Visual Studio 2017**, операційна система **Windows 10**; веб-браузер **Google Chrome** для роботи з веб-сайтами для пошуку необхідної інформації; текстовий редактор **Microsoft Word 2010** для підготовки та оформлення курсової роботи.

Робота складається зі вступу, трьох розділів, висновків, додатків та списку використаних джерел.

**РОЗДІЛ 1**

**ПОСТАНОВКА ЗАДАЧІ**

Постановка задачі полягає у реалізації синтаксичного аналізу введеного користувачем рядка для перетворення його на математичну функцію та побудові графіку цієї функції.

* 1. **Огляд існуючих підходів до розв’язання поставленої задачі**

Існує багато алгоритмів аналізу виразів («метод з двома стеками», «рекурсивного спуску», «алгоритм сортувальної станції» тощо). В даній роботі реалізований алгоритм «рекурсивного спуску».

Для побудови графіку та створення сприятливого інтерфейсу використані функції, передбачені у стандартних бібліотеках.

* 1. **Уточнена постановка задачі на розробку програмного забезпечення**

Розробка програми проходить в два етапи. Перший етап передбачає реалізацію парсингу математичного виразу за допомогою алгоритму «рекурсивного спуску».

Суть даного методу полягає в тому, що він поділяється на свої підзадачі. У свою чергу підзадача повинна працювати тільки з тим, з чим вміє працювати, якщо умови не задовольняються передавати управління далі. Якщо ж умови задоволені, робимо обчислення і передаємо частину необробленого тексту. Виконання відбувається до тих пір, поки текст все ще є або ж якщо жодна підзадача не може обробити поточний стан. Також має значення пріоритет підзадачі. Змінюючи пріоритет підзадачі зміниться і поведінка парсера.

Оскільки реалізовувався парсер математичних формул, то для розставляння пріоритетів керувалися пріоритетами математичних операцій: функція і змінна, дужки, множення і ділення, додавання і віднімання.

Вхідний вираз відповідає припустимому опису, якщо:

* Кількість дужок « ( » дорівнює кількості дужок « ) ».
* Ціла частина числа відокремлена від дробової за допомогою точки.
* У рядку присутні тільки допустимі символи: цифри 0 ... 9, оператори «+», «-», «\*», «/», «%», «x^(…)», «sin(...)», «cos(…)», «tan(...)», «ctan(…)», дужки, крапка і параметр x.

Другий етап полягає у побудові графіку. Він реалізований за допомогою функції, яка визначає значення ординати для кожного аргументу та передає ці координати іншій функції, що містить графічні методи.

**РОЗДІЛ 2**

**РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ**

* 1. **Метод розв’язку задачі**

Реалізований синтаксичний аналізатор проходить у три етапи: зчитування рядка, обчислення результату, повернення результату. Спочатку отриманий від користувача рядок розбивається на лексеми. Лексемами можуть бути числа з плаваючою крапкою, дозволені оператори та скобки. При цьому граматика виразу має такий вигляд:

Вираз:

Терм

  Вираз «+» Терм

Вираз «-» Терм

Терм:

  Первинний вираз

  Терм «\*» Первинний вираз

  Терм «/» Первинний вираз

  Терм «%» Первинний вираз

Терм «^» Первинний вираз

Первинний вираз:

  Число, змінна

  «(» Вираз «)»

«sin», «cos», «tan», «ctan»

Число:

  літерал\_с\_плаваючою\_крапкою

Лексеми можемо поділити на два види: числа та всі інші. Для реалізації цієї ідеї створено клас Token, що містить поля вид (kind) та значення (value), а також рядкове поле роботи з функціями, які містять більше одного символу. Всі лексеми, окрім чисел мають порожнє поле зі значенням, а в полі вид записано саму лексему. Якщо лексема є функцією типу «sin», «cos», «tan», «ctan», то в поле вид записується сигнальний символ, а в рядкове поле назва (name) відповідна лексема. Якщо лексема є числом, то в поле вид записується сигнальний символ, а в поле значення відповідне число.

Для роботи з лексемами реалізовані функція для зчитування лексем та функції для кожного граматичного правила.

Для встановлення розміру консолі та побудови графіку наданої функції використовуються графічні методи зі стандартної бібліотеки «windows.h».

* 1. **Алгоритм розв’язку задачі**

Робота програми починається з введення користувачем бажаної функції. Користувач бачить короткі пояснення роботи програми та ескіз системи координат, в якій буде побудовано майбутній графік. Для того, щоб побачити графік введеної функції, користувач має ввести символ «;». Якщо ця функція має неприпустимий опис, то виводиться повідомлення про помилку і виконання програми завершується. В іншому випадку користувач побачить бажаний графік та матиме вибір: припинити виконання програми чи побудувати новий графік.

Після зчитування рядка викликається функція для побудови графіку graph(…), яка в свою чергу викликає функцію для обчислення значень координат calculate(…). Остання викликає функцію expression(…), з якої починається рекурсія, як показано на рисунку 2.1.

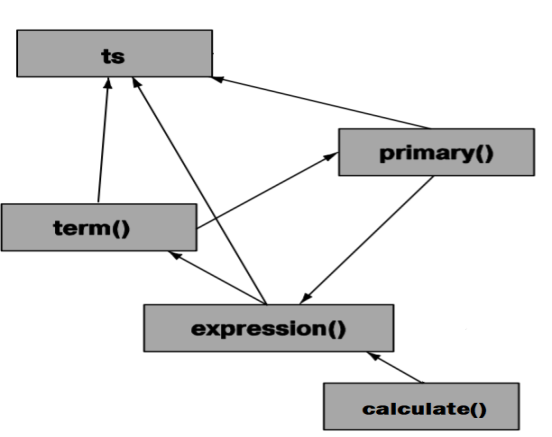


Рис. 2.1 Рекурсія для обчислення значення виразу

( **ts** – об‘єкт класу TokStr)

Для роботи з лексемами реалізовано клас TokStr, який містить поле буферу для збереження лексеми та об‘єкт потоку stringstream, в який передається введений користувачем рядок. Лексема зчитується з цього потоку та повертається для подальшого опрацювання. Якщо лексема не була використана для обчислень, то вона тимчасово поміщається в буфер, з якого буде вилучена при наступному виклику функції для взяття лексеми.

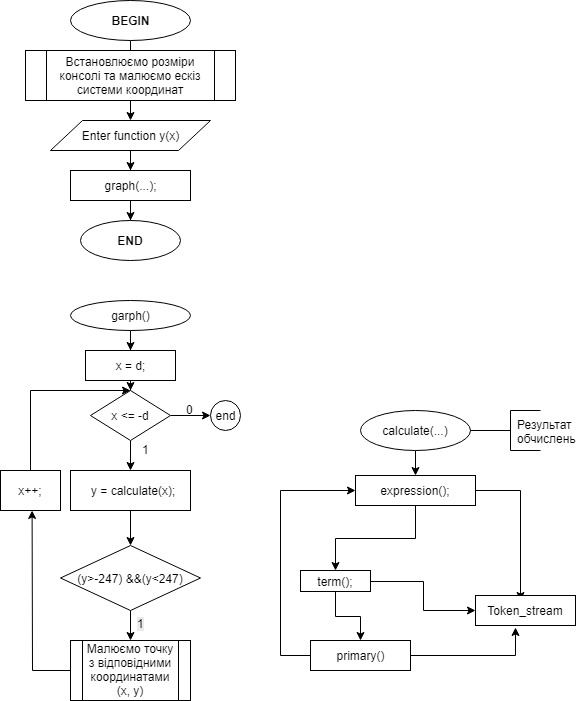


Рис. 2.2 Структурні блок-схеми головних функціональних блоків

**РОЗДІЛ 3**

**ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ**

Програма складається з основного та двох заголовкових файлів. Перший заголовковий клас містить реалізацію двох класів Token та TokStr. Другий заголовковий файл містить функції для обчислення значення виразу для певного аргументу. В основному файлі містяться графічні методи для встановлення розміру консолі та побудови графіку.

**3.1 Опис головних структур і змінних програми**

Таблиця 3.1 Опис головних структур і змінних програми.

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Змінна** | **Тип змінної** | **Призначення змінної** |
| class Token | | | |
| 1 | kind | char | Містить лексему або вказує на вид лексеми (якщо, наприклад, це число). |
|
| 2 | value | double | Містить значення лексеми, якщо вона є числом, тобто саме число. |
|
| 3 | name | string | Містить назву функції, якщо вона складається більш ніж з одного символу. |
|
| class TokStr | | | |
| 4 | full | bool | Визначає, чи буфер повний. |
|
| **№** | **Змінна** | **Тип змінної** | **Призначення змінної** |
| 5 | buffer | Token | Тимчасово зберігає лексему, якщо вона не була використана. |
| 6 | ss | stringstream | Об`єкт потоку, що дозволяє зчитувати лексеми із рядка. |
|  | | | |
| 7 | key | int | Змінна, що зберігає код натиснутої клавіши. |
| 8 | ts | TokStr | Обєкт класу, що дозволяє працювати з лексемами. За потреби зберігає лексему поки вона не буде використана. |
|
| 9 | s | string | Рядок, який містить функцію користувача. |
|
|  | | | |
| 10 | i1 | int | Ліва координата поля для графіку по х. |
|
| 11 | i2 | int | Верхня координата поля для графіку по у. |
|
| 12 | i3 | int | Права координата поля для графіку по х. |
|
| 13 | i4 | int | Нижня координата поля для графіку по у. |
|
| **№** | **Змінна** | **Тип змінної** | **Призначення змінної** |
| 14 | х | double | Поточна координата по х |
| 15 | у | double | Поточна координата по y |
|

**3.2 Опис головних функцій програми**

Таблиця 3.2. Опис головних функцій програми.

|  |  |  |
| --- | --- | --- |
| **№** | **Синтаксис** | **Опис** |
| class Token | | |
| 1 | Token(char ch) | Конструктор класу з параметром. Встановлює вид лексеми (саму лексему). |
|
| 2 | Token(char ch, double val) | Конструктор класу з параметрами. Встановлює вид лексеми та її значення (число). |
|
| 3 | Token(char ch, string n) | Конструктор класу з параетрами. Встановлює вид лексеми та назву лексеми (операції). |
| 4 | ~Token() | Деструктор класу. |
| **№** | **Синтаксис** | **Опис** |
| class TokStr | | |
| 5 | TokStr() | Конструктор умовчання класу. Створює об`єкт. |
| 6 | void Set(string f) | Встановлює рядок, з якого будуть зчитуватись лексеми. |
| 7 | Token get() | Зчитує посимвольно з рядка ss, визначає лексему та повертає її, скориставшись відповідним конструктором класу Token. |
| 8 | void putback(Token t) | Функція заносить лексему, якою ще не скористались, до буферу. |
| 9 | ~TokStr() | Деструктор класу. |
| **№** | **Синтаксис** | **Опис** |
| 10 | double calculate(double z); | Функція для обчислення значення ординати для відповідного значення аргументу. Аргумент передається до функції як параметр. |
| 11 | double expression(double z); | Опрацьовує операції «+», «-» . |
| 12 | double term(double z); | Опрацьовує операції «\*», «/», «%», «x^(…)». |
| 13 | double primary(double z); | Опрацьовує операції «sin(...)», «cos(…)», «tan(...)», «ctan(…)», дужки, унарний мінус та плюс, аргумент. |
|  | | |
| 14 | void draw(HWND hwnd, HDC hdc, int i1 , int i2 , int i3, int i4 ); | Малює поле з системою координат, де буде побудовано графік. |
|
| 15 | void graph(HWND hwnd, HDC hdc, int i1, int i2, int i3, int i4); | Малює графік функції, що ввів користувач. |
|
| 16 | void setConsoleSize(); | Встановлює розмір консолі. |
|

**3.3 Опис інтерфейсу**

Користувач бачить перед собою видозмінене консольне вікно, в якому містяться короткі вказівки щодо роботи програми та поле з системою координат, на якому буде побудовано майбутній графік, як показано на рисунку 3.1.

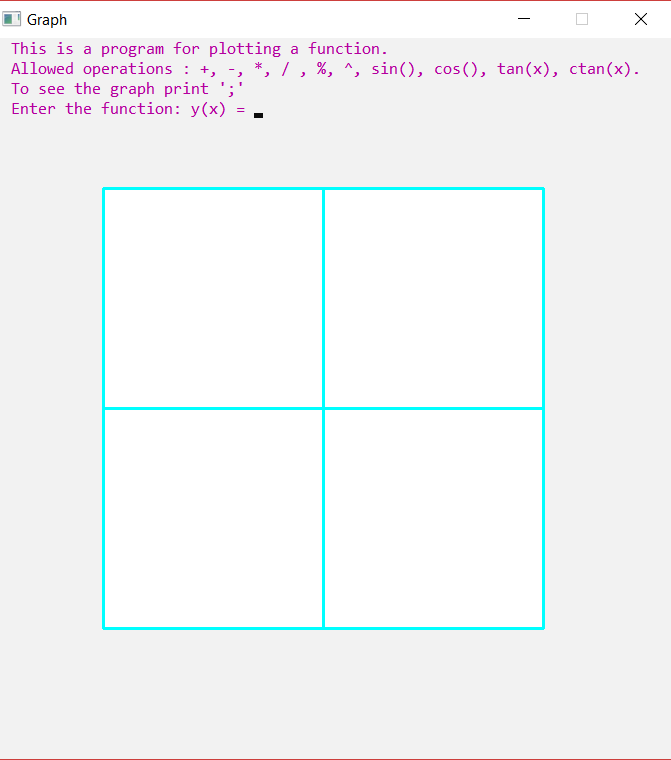


Рис.3.1. Вигляд консольного вікна на початку роботи програми

Користувачу пропонується ввести бажану функцію y(x). Для того, щоб користувач побачив графік, в кінці він має надрукувати символ «;», який є сигнальним.

Після того, як графік побудовано, перед користувачем постає вибір: побудувати новий графік чи завершити роботу.

**3.4 Результати роботи програмного продукту.**

Результати роботи програми зображено на рисунках 3.2 – 3.7.

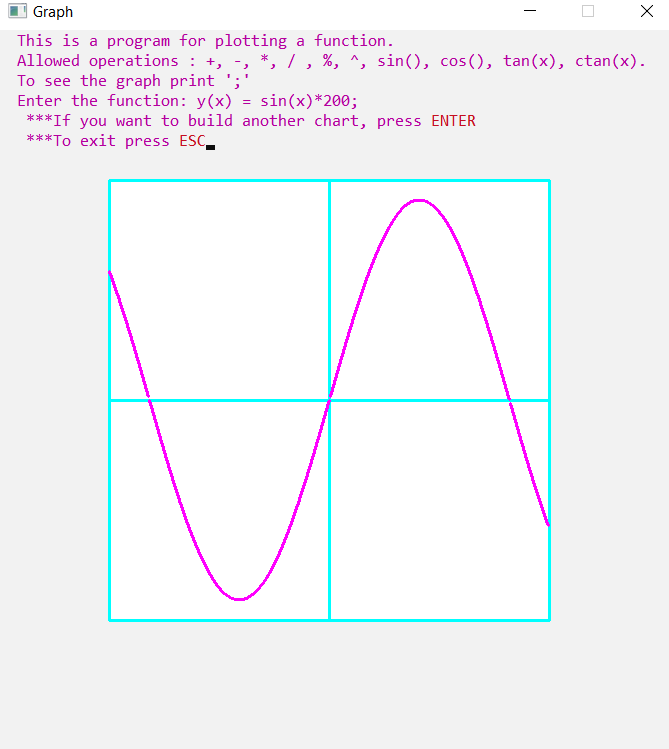


Рис.3.2. Результат роботи програми для функції y(x) = sin(x)\*200.

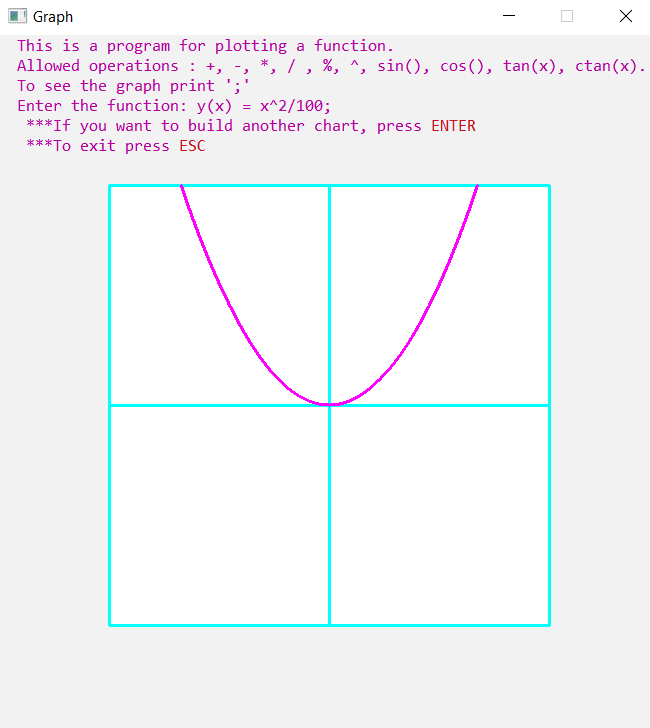


Рис.3.3. Результат роботи програми для функції y(x) = x^2/100.

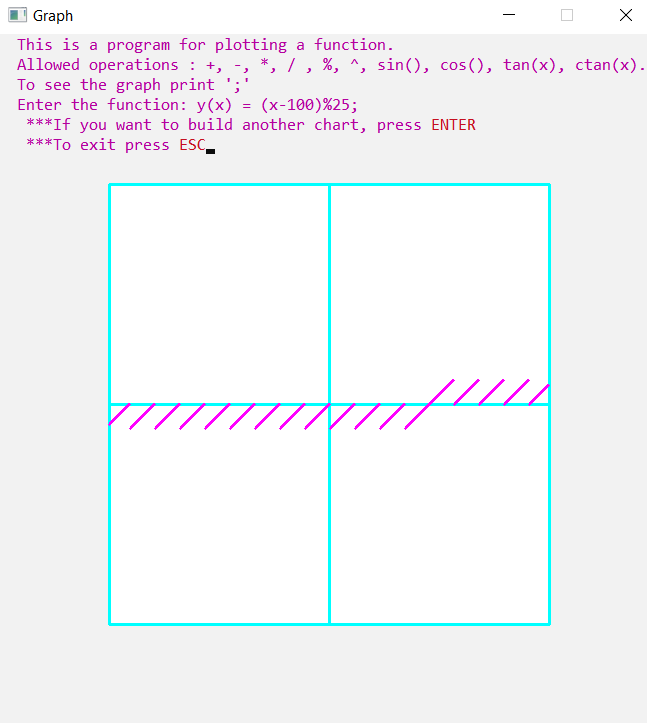


Рис.3.4. Результат роботи програми для функції y(x) = (x-100)%25.

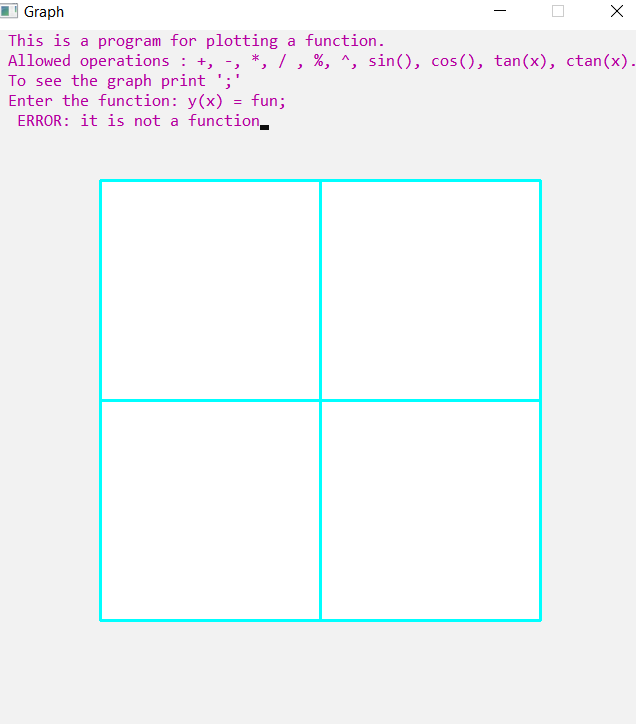


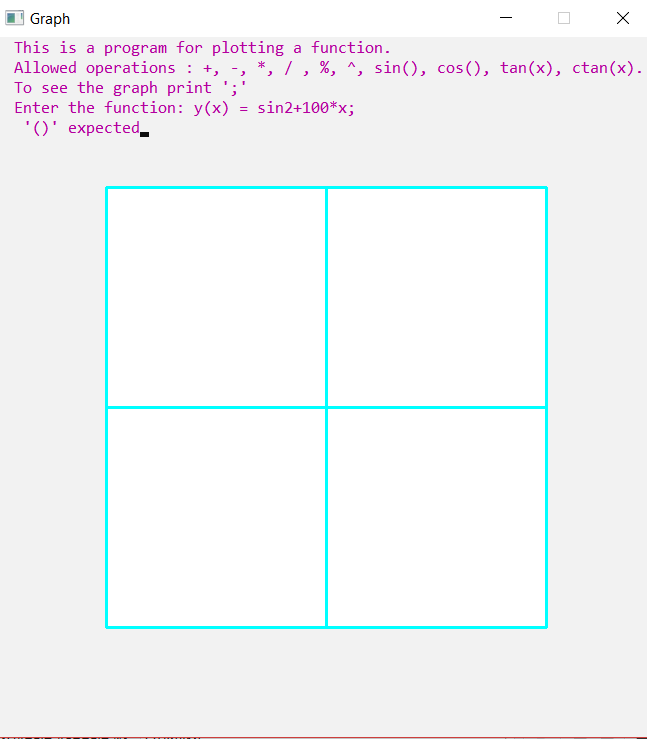
Рис.3.5. Виведення про помилку (некоректні вхідні дані). 

Рис.3.6.Виведення про помилку (відсутні дужки).

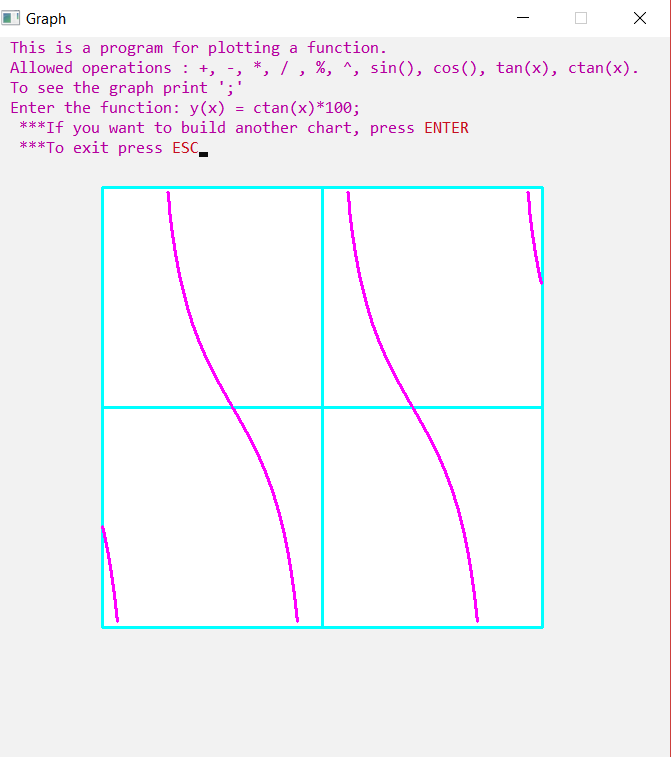


Рис.3.7.Робота програми для функції y(x) = ctan(x)\*100.

Програма працює та будує графіки, що дійсно відповідають введеним даним. Недоліком є те, що при малих значеннях констант графік погано видно. Для того, щоб добре бачити графік мусимо множити або ділити на велике число.

**ВИСНОВКИ**

Під час розробки програмного продукту вдалося повністю реалізувати поставлене завдання побудови графіку функції з використанням синтаксичного аналізатору. Дана програма може стати допоміжним засобом для розв‘язання деяких математичних та фізичних задач.

За допомогою об‘єктно-орієнтованої мови програмування С++, використанням стандартних бібліотек вдалося реалізувати синтаксичний аналіз математичного виразу та побудову відповідного графіку з використанням графічних функцій, що містяться в бібліотеці «windows.h».

Основним середовищем розробки, яке використовувалось при розробці програмного продукту є «Visual Studio 2017».

Головним недоліком програми є обмеження щодо побудови графіку. По-перше, неможливість розгледіти графік при малих значеннях констант. По-друге, маємо невеликий набір операцій, які розпізнає програма.

Перевагою програми є можливість багаторазово будувати графіки, не припиняючи роботу. Також позитивним аспектом є відсутність зайвої інформації, що заважає сприймати адекватно роботу програмного продукту. Кольорове оформлення справляє приємне враження і робить інтерфейс більш дружнім до користувача.

Отже, програма працює та виконує поставлену задачу, але може бути удосконалена. Планую для покращення в майбутньому даного програмного продукту розширити вибір операцій для побудови графіку, наприклад log(), arcsin(), exp() тощо.

**Список використаних джерел**

* 1. Страуструп, Бьерне. Программирование: принципы и практика использования C++ , испр. узд. : Пер. с англ. – М. :ООО «И.Д.Вильямс», 2011. – 1248 с. : ил. – Парал. тит. англ.
  2. Хортон, Айвор. Visual С++ 2010: полный курс. : Пер. с англ. – М.: ООО «И.Д. Вильямс», 2011. – 1216 с. : ил. – Парал. тит. англ..
  3. MSDN - Windows API Персональный сайт Владимира Соковикова (<http://vsokovikov.narod.ru/>)

1. Програмування: Метод. вказівки до самостійної роботи студентів напряму підготовки 6.050101 «Комп’ютерні науки» / Уклад.: І.В. Назарчук, О.Л. Тимощук – К. ННК «ІПСА» НТУУ «КПІ», 2012. – 23 с.
2. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення: ДСТУ 3008-95 (ГОСТ 7.32-91). – Чинний від 2006-01-01. К.: Держспоживстандарт України, 2006.– 231с. – (Національний стандарт України).
3. ДСТУ ГОСТ 7.1.2006 «Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання».
   1. ДСТУ ГОСТ 7.1:2006 библиографический список, примеры оформления//Известия вузов. Радиоэлектроника. — Режим доступа:<http://radio.kpi.ua/author/rules/11-author-new-biblio>.
4. Програмування та алгоритмічні мови 2. Програмування:Методичні вказівки до виконання курсового проекту для студентів галузі знань 12 «Інформаційні технології» спеціальність 122 «Комп’ютерні науки та інформаційні технології»,

124 «Системний аналіз»/Уклад.:І.В.Назарчук, Г.Г.Швачко – К.НТУУ «КПІ», 2017 – 35с.

**Додаток. Текст програми**

*Основна частина*

#include "stdafx.h"

#include "TOKEN.h"

#include <windows.h>

#define \_WIN32\_WINNT 0x0500

using namespace std;

void draw(HWND hwnd, HDC hdc, int i1 = 110, int i2 = 150, int i3 = 550, int i4 = 590);

void graph(HWND hwnd, HDC hdc, int i1 = 110, int i2 = 150, int i3 = 550, int i4 = 590);

void setConsoleSize();

int key = 1;

int main(){

system("color F0");

HANDLE h = GetStdHandle(STD\_OUTPUT\_HANDLE);

HWND hwnd = GetConsoleWindow();

HDC hdc = GetDC(hwnd);

setConsoleSize();

SetConsoleTitle(L"Graph");

SetWindowLong(hwnd, GWL\_STYLE, GetWindowLong(hwnd, GWL\_STYLE) & ~WS\_MAXIMIZEBOX & ~WS\_SIZEBOX);

SetConsoleTextAttribute(h, BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_RED | BACKGROUND\_INTENSITY |FOREGROUND\_INTENSITY | 13);

SetBkColor(hdc, RGB(0, 255, 255));

Sleep(100);

draw(hwnd, hdc);

cout << " This is a program for plotting a function."<<endl;

cout << " Allowed operations : +, -, \*, / , %, ^, sin(), cos(), tan(x), ctan(x)."<<endl;

cout << " To see the graph print ';'" << endl;

cout << " Enter the function: y(x) = ";

getline(cin, s);

graph(hwnd, hdc);

cout << " \*\*\*If you want to build another chart, press ";

SetConsoleTextAttribute(h, BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_RED | BACKGROUND\_INTENSITY | 4); cout << "ENTER" << endl;

SetConsoleTextAttribute(h, BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_RED | BACKGROUND\_INTENSITY | FOREGROUND\_INTENSITY | 13);

cout << " \*\*\*To exit press ";

SetConsoleTextAttribute(h,BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_RED | BACKGROUND\_INTENSITY| 4); cout << "ESC";

while (key != 13 || key != 27) {

if (\_kbhit()) {

key = \_getch();

if (key == 27) exit(0);

if (key == 13) {

system("cls");

main();

}

}

}

\_getch();

ReleaseDC(NULL, hdc);

return 0;

}

void draw(HWND hwnd, HDC hdc, int i1 , int i2 , int i3, int i4 ) {

Rectangle(hdc, i1, i2, i3, i4);

HPEN Pen = CreatePen(PS\_SOLID, 3, RGB(0, 255, 255));

SelectObject(hdc, Pen);

MoveToEx(hdc, i1, i2, NULL);

LineTo(hdc, i3, i2);

LineTo(hdc, i3, i4);

LineTo(hdc, i1, i4);

LineTo(hdc, i1, i2);

MoveToEx(hdc, (i1 + i3) / 2, i2, NULL);

LineTo(hdc, (i1 + i3) / 2, i4);

MoveToEx(hdc, i1, (i2+i4)/2, NULL);

LineTo(hdc, i3, (i2 + i4) / 2);

DeleteObject(Pen);

};

void graph(HWND hwnd, HDC hdc, int i1, int i2, int i3, int i4) {

double x , y = -228, z;

int a = (i1 + i3) / 2, b = (i4 + i2) / 2, c = (i4 - i2) / 2, d = (i1 - i3) / 2;

HPEN Pen = CreatePen(PS\_SOLID, 3, RGB(255, 0, 255));

SelectObject(hdc, Pen);

x = d;

while (y <= b - i4 || y >= b - i2) {

y = calculate(x);

MoveToEx(hdc, x + a, b - y, NULL);

x++;

} z = y;

for (; x < -d; x+=0.1) {

y = calculate(x);

while ((y <= b - i4 || y >= b - i2) && (x<-d)) {

y = calculate(x);

MoveToEx(hdc, x + a, b - y, NULL);

x++;

}

if ((z >= 0 && y < 0) || (z > 0 && y<=0) || (z <= 0 && y > 0) || (z < 0 && y >= 0)) MoveToEx(hdc, x + a, b - y, NULL);

if (y > b-i4 && y < b-i2) {

LineTo(hdc, (int)x + a, b - y);

}

z = y;

};

}

void setConsoleSize() {

const int colConsole = 73;

const int rowConsole = 35;

HANDLE hNdl = GetStdHandle(STD\_OUTPUT\_HANDLE);

SMALL\_RECT windowSize = { 0,0,colConsole - 1, rowConsole - 1 };

SetConsoleWindowInfo(hNdl, TRUE, &windowSize);

COORD bufferSize = { colConsole, rowConsole };

SetConsoleScreenBufferSize(hNdl, bufferSize);

}

*Заголовковий файл TOKEN.h*

#include "TOKENSTREAM.h"

double expression(double z);

double term(double z);

double primary(double z);

double calculate(double z);

TokStr ts;

string s;

const double Pi = 3.141592;

double expression(double z) { // +,-

double left = term(z);

Token t = ts.get();

while (true) {

switch (t.kind) {

case '+': {

left += term(z);

t = ts.get();

break;

}

case '-': {

left -= term(z);

t = ts.get();

break;

}

default:

ts.putback(t);

return left;

}

}

}

double term(double z) { //\*,/

double left = primary(z);

Token t = ts.get();

while (true) {

switch (t.kind) {

case '\*': {

left \*= primary(z);

t = ts.get();

break;

}

case '/': {

double d = primary(z);

left /= d;

t = ts.get();

break;

}

case '%': {

double d = primary(z);

int i1 = (int)left;

int i2 = (int)d;

left = i1 % i2;

t = ts.get();

break;

}

case '^': {

double d = primary(z);

if (d < 1) {

int k = 1 / d;

if (k % 2 != 0) {

if (left < 0) {

left = - pow(-left, d);

}

else left = pow(left, d);

}

else {

if (left < 0) {

left = -228;

} else left = pow(left, d);

}

} else left = pow(left, d);

t = ts.get();

break; }

}

double primary(double z) {//(1234)

Token t = ts.get();

switch (t.kind) {

case '(': {

double d = expression(z);

t = ts.get();

if (t.kind != ')') {

cout << (" ')'expected");

\_getch(); exit(2);

}

return d;

}

case func: {

Token q = ts.get();

if (q.kind != '(') {

cout << " '()' expected";

\_getch(); exit(2);

}

else ts.putback(q);

double d = primary(z);

if (t.func == "sin")

return sin(Pi\*d/180);

if (t.func == "cos")

return cos(d\*Pi/180);

if (t.func == "tan")

return tan(d\*Pi / 180);

if (t.func == "ctan")

return 1/tan(d\*Pi / 180);

}

case number: return t.value;

case 'x': return z;

case '-': return -primary(z);

case '+': return primary(z);

default: {

cout << (" Expression was expected");

\_getch(); exit(2); }

}

}

double calculate(double z) {

ts.Set(s);

Token t = ts.get();

while (t.kind == plot) t = ts.get();

ts.putback(t);

return expression(z);

}

*Заголовковий файл TOKENSTREAM.h*

#include <sstream>

#include <stdio.h>

#include <math.h>

#include <iostream>

#include <string.h>

#include <conio.h>

using namespace std;

const char number = '$';

const char plot = ';';

const char func = '!';

class Token {

public:

char kind;

double value;

string func;

Token(char ch) :kind(ch), value(0) { }

Token(char ch, double val) :kind(ch), value(val) { }

Token(char ch, string n) :kind(ch), func(n) {}

~Token() { }

};

class TokStr {

private:

bool full;

Token buffer;

stringstream ss;

public:

TokStr() :full(false), buffer(0) {}

void Set(string f) { ss << f; }

Token get() {

if (full) {

full = false;

return buffer;

}

char ch;

ss >> ch;

switch (ch) {

case plot:

case '(':

case ')':

case '^':

case '+':

case '-':

case '\*':

case '/':

case '%':

{return Token(ch); }

case '.':

case '0': case '1': case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '9': {

ss.putback(ch);

double val;

ss >> val;

ss >> ch;

if (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^' || ch == '%' || ch == ')' || ch == ';') {

ss.putback(ch);

return Token(number, val);

}

else {

cout << (" ERROR: the operator is omitted after val"); \_getch(); exit(2);

}

break;

}

default:

if (isalpha(ch)) {

if (ch == 'x') {

ss.get(ch);

if (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^' || ch == '%' || ch == ')' || ch == ';')

{

ss.putback(ch);

return Token('x');

}

else {

cout << (" ERROR: the operator is omitted after var"); \_getch(); exit(2);

}

}

string s;

s += ch;

while (ss.get(ch) && isalpha(ch))

s += ch;

ss.putback(ch);

if (s != "sin" && s != "cos" && s!="tan" && s!="ctan") {

cout << " ERROR: it is not a function";

\_getch(); exit(2);

} else return Token(func, s);

}

cout << (" ERROR: function does not have a feasible description "); \_getch(); exit(2);

}

}

void putback(Token t) {

if (full) cout << ("putback() to full buffer");

buffer = t;

full = true;

}

~TokStr(){}

};