

**Assalamualaikum wr wb.**

Kami dari Tim tryingshappy beranggotakan:

Decya Giovanni	5025221027
Pelangi Masita Wati	5025221051
Muhammad Abdurrahman Faiz	5025221058
Mohammad Hanif Furqan Aufa Putra	5025221161

## **TUGAS FINAL PROJECT**

### **KELAS PBO C**

PLOT1 - Plotting functions (variation)

*no tags*

Given a function  $y=f(x)$  in RPN-notation plot it with stars (\*) and then (!) its derivation with crosses (+) for  $0 \leq x \leq 20$  with  $\Delta x=1$  in a diagram with  $21 \times 21$  points ( $0 \leq x, y \leq 20$ ). Empty fields are marked with dots (.). For plotting the real number  $y$  should be rounded to integer ( $-0.5 \rightarrow -1$ ,  $-0.4 \rightarrow 0$ ,  $0.4 \rightarrow 0$ ,  $0.5 \rightarrow 1$ ). The function and its derivation are continuous between 0 and 20.

The function definition uses only the following characters: 0123456789x.+\*/^  
'^' means 'power of'. Items are separated by space.

Input

In the first line the number  $N$  of functions, then  $N$  lines with one function.

Output

The plot of each function and its derivation in 21 lines.

Example

Input:

1

x 1 -

Output:

.....

.....\*



1. Evaluasi fungsi pada setiap nilai  $x$  dari 0 hingga 20 dengan langkah  $\Delta x=1$ .
2. Hasil evaluasi fungsi dibulatkan ke bilangan bulat sesuai aturan tertentu.
3. Plot fungsi dengan menandai nilai  $y$  yang sesuai menggunakan karakter '\*' dalam matriks  $21 \times 21$ .
4. Turunkan fungsi.
5. Evaluasi turunan pada setiap nilai  $x$  dari 0 hingga 20 dengan langkah  $\Delta x=1$ .
6. Hasil evaluasi turunan dibulatkan ke bilangan bulat sesuai aturan tertentu.
7. Plot turunan dengan menandai nilai  $y$  yang sesuai menggunakan karakter '+' dalam matriks yang sama.

Hasil akhirnya adalah matriks  $21 \times 21$  yang mencerminkan plot dari setiap fungsi dan turunan fungsi. Posisi kosong dalam matriks ditunjukkan oleh karakter '.'.

Pendekatan yang digunakan untuk menyelesaikan masalah ini dapat diuraikan sebagai berikut:

1. Kelas fungsipola: Dalam kelas ini, dibuat objek untuk setiap fungsi yang dinyatakan dalam Notasi Polandia Terbalik (RPN). Konstruktor menerima ekspresi RPN dan menyimpannya. Terdapat metode plotting yang digunakan untuk mengevaluasi fungsi pada setiap nilai  $x$  dari 0 hingga 20 dan menggambar plot fungsi dan turunannya pada matriks  $21 \times 21$ .
2. Metode pola: Metode ini menggunakan loop untuk iterasi melalui nilai  $x$  dari 0 hingga 20. Pada setiap iterasi, nilai fungsi dan turunan dihitung menggunakan metode calcRPN dan calcTurunan. Hasil perhitungan tersebut dibulatkan sesuai dengan aturan yang diberikan dan kemudian posisi yang sesuai dalam matriks  $21 \times 21$  ditandai dengan karakter '\*' untuk fungsi dan '+' untuk turunan.
3. Metode calcRPN: Metode ini mengimplementasikan evaluasi ekspresi RPN menggunakan stack. Setiap token dalam ekspresi diproses, dan hasilnya dihitung dengan memanipulasi stack sesuai dengan operator dan operand yang ditemukan.
4. Metode calcTurunan: Metode ini menghitung turunan numerik dengan menggunakan rumus beda hingga. Nilai fungsi dievaluasi pada titik  $x+h$  dan  $x-h$ , kemudian turunan dihitung sebagai perbedaan nilai fungsi dibagi dengan dua kali nilai  $h$ .
5. Metode setGridPoint: Metode ini memastikan bahwa nilai  $y$  yang akan ditandai pada matriks berada dalam rentang yang valid (antara 0 dan gridSize). Ini membantu menghindari kesalahan saat menetapkan karakter pada matriks.
6. Fungsi main: Di fungsi utama, dibaca jumlah fungsi ( $N$ ) dari input, kemudian membaca setiap ekspresi fungsi. Untuk setiap fungsi, dibuat objek fungsipola, menginisialisasi matriks grid dengan karakter '.', dan kemudian memanggil metode plotting. Setelah itu, matriks hasil dioutputkan ke layar dengan membalikkan urutan baris agar tampilan sesuai dengan spesifikasi yang diminta.

## Kode Program:

```
#include <stdio>
#include <cmath>
#include <stack>
#include <string>
#include <string>

constexpr int gridSize = 21;
constexpr int maxExpressionSize = 1000;

class fungsipola
{
public:
    fungsipola(const char *rpn) : rpn(rpn) {}
    void pola(char polaGrid[gridSize][gridSize])
    {
        for (int x = 0; x < gridSize; ++x)
        {
            double value = cacLRPN(x);
            double derivative = calcTurunan(x);

            setGridPoint(polaGrid, x, nilaibulat(value), '*');
            setGridPoint(polaGrid, x, nilaibulat(derivative), '+');
        }
    }

    void cetak(char polaGrid[gridSize][gridSize])
    {
        for (int y = gridSize - 1; y >= 0; --y)
        {
            for (int x = 0; x < gridSize; ++x)
            {
                printf("%c", polaGrid[y][x]);
            }
            printf("\n");
        }
    }
}
```

```
private:
    const char *rpn;

    int nilaibulat(double value)
    {
        return static_cast<int>(value + (value < 0 ? -0.5 : 0.5));
    }

    double cacLRPN(double x)
    {
        std::stack<double> RPNstack;
        char expression[maxExpressionSize];
        strcpy(expression, rpn);
        char *token = strtok(expression, " ");

        while (token != NULL)
        {
            if (isdigit((unsigned char)token[0]) || (token[0] == '-' && isdigit((unsigned char)token[1])))
            {
                RPNstack.push(atof(token));
            }
            else if (strcmp(token, "x") == 0)
            {
                RPNstack.push(x);
            }
            else
            {
                performOperation(token, RPNstack);
            }

            token = strtok(NULL, " ");
        }

        return RPNstack.top();
    }

    double calcTurunan(double x)
    {
        const double h = 0.0001;
        return (cacLRPN(x + h) - cacLRPN(x - h)) / (2 * h);
    }

    void setGridPoint(char polaGrid[gridSize][gridSize], int x, int y, char symbol)
    {
        if (y >= 0 && y < gridSize)
        {
            polaGrid[y][x] = symbol;
        }
    }
}
```

```
void performOperation(const char *token, std::stack<double> &RPNstack)
{
    double operand2 = RPNstack.top();
    RPNstack.pop();
    double operand1 = RPNstack.top();
    RPNstack.pop();

    if (strcmp(token, "+") == 0)
    {
        RPNstack.push(operand1 + operand2);
    }
    else if (strcmp(token, "-") == 0)
    {
        RPNstack.push(operand1 - operand2);
    }
    else if (strcmp(token, "*") == 0)
    {
        RPNstack.push(operand1 * operand2);
    }
    else if (strcmp(token, "/") == 0)
    {
        RPNstack.push(operand1 / operand2);
    }
    else if (strcmp(token, "^") == 0)
    {
        RPNstack.push(pow(operand1, operand2));
    }
    else
    {
        RPNstack.push(0.0);
    }
}

};
```

```
int main()
{
    int functions;
    scanf("%d", &functions);

    int c;
    while ((c = getchar()) != '\n' && c != EOF)
    {
    }

    char grid[gridSize][gridSize];

    for (int i = 0; i < functions; ++i)
    {
        char expression[maxExpressionSize];
        scanf("%999[^\n]", expression);
        getchar();

        fungsipola plotter(expression);

        for (int y = 0; y < gridSize; ++y)
        {
            for (int x = 0; x < gridSize; ++x)
            {
                grid[y][x] = '.';
            }
        }

        plotter.pola(grid);
        plotter.cetak(grid);
    }

    return 0;
}
```

Penjelasan:

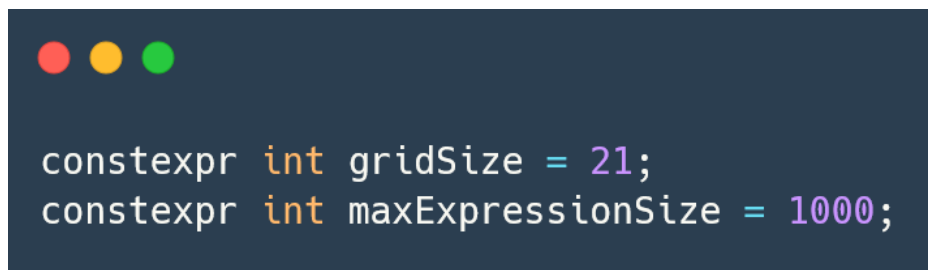
1. Library yang digunakan



```
#include <cstdio>
#include <cmath>
#include <stack>
#include <cstring>
#include <string>
```

- a) `cstdio`: Library ini digunakan untuk input dan output standar. Dalam kode ini, digunakan untuk membaca input dari pengguna dan menampilkan output ke layar.
- b) `string`: Library ini menyediakan berbagai fungsi dan objek untuk memanipulasi string. Dalam kode ini, digunakan untuk memanipulasi ekspresi matematika yang dinyatakan dalam RPN.
- c) `cmath`: Library ini menyediakan fungsi-fungsi matematika yang melibatkan operasi-operasi kompleks. Dalam kode ini, digunakan untuk menghitung nilai-nilai matematika seperti akar kuadrat dan pangkat.
- d) `stack`: Library ini menyediakan struktur data stack, yang digunakan dalam implementasi evaluasi ekspresi RPN. Stack digunakan untuk menyimpan operand dan operator sementara selama proses evaluasi.
- e) `cstring`: Library ini menyediakan fungsi-fungsi untuk manipulasi string dan karakter. Dalam kode ini, digunakan untuk berbagai operasi terkait string

2. Deklarasi dan penggunaan deklarasi variabel konstan



```
constexpr int gridSize = 21;
constexpr int maxExpressionSize = 1000;
```

Baris ini mengindikasikan penggunaan namespace std dan mendeklarasikan variabel konstan gridSize dengan nilai 21. Kata kunci constexpr menandakan bahwa nilai variabel ini akan dievaluasi selama kompilasi dan tidak dapat diubah setelahnya. Dalam kode tersebut, maxExpressionSize adalah sebuah konstanta yang dideklarasikan sebagai constexpr int dengan nilai 1000. Konstanta ini digunakan sebagai ukuran maksimal buffer atau array untuk menyimpan ekspresi fungsi yang dibaca dari input. Dengan kata lain, panjang maksimum string ekspresi fungsi yang dapat diakomodasi oleh buffer ini adalah 1000 karakter. Konstanta ini membantu dalam menghindari buffer overflow atau masalah ukuran buffer yang tidak mencukupi saat membaca ekspresi fungsi.

### 3. Kelas fungsipola

```
class fungsipola
{
public:
    fungsipola(const char *rpn) : rpn(rpn) {}
    void pola(char polaGrid[gridSize][gridSize])
    {
        for (int x = 0; x < gridSize; ++x)
        {
            double value = cacLRPN(x);
            double derivative = calcTurunan(x);

            setGridPoint(polaGrid, x, nilaibulat(value), '*');
            setGridPoint(polaGrid, x, nilaibulat(derivative), '+');
        }
    }

    void cetak(char polaGrid[gridSize][gridSize])
    {
        for (int y = gridSize - 1; y >= 0; --y)
        {
            for (int x = 0; x < gridSize; ++x)
            {
                printf("%c", polaGrid[y][x]);
            }
            printf("\n");
        }
    }
};
```

```
private:
    const char *rpn;

    int nilaibulat(double value)
    {
        return static_cast<int>(value + (value < 0 ? -0.5 : 0.5));
    }

    double cacLRPN(double x)
    {
        std::stack<double> RPNstack;
        char expression[maxExpressionSize];
        strcpy(expression, rpn);
        char *token = strtok(expression, " ");

        while (token != NULL)
        {
            if (isdigit((unsigned char)token[0]) || (token[0] == '-' && isdigit((unsigned char)token[1])))
            {
                RPNstack.push(atof(token));
            }
            else if (strcmp(token, "x") == 0)
            {
                RPNstack.push(x);
            }
            else
            {
                performOperation(token, RPNstack);
            }

            token = strtok(NULL, " ");
        }

        return RPNstack.top();
    }

    double calcTurunan(double x)
    {
        const double h = 0.0001;
        return (cacLRPN(x + h) - cacLRPN(x - h)) / (2 * h);
    }

    void setGridPoint(char polaGrid[gridSize][gridSize], int x, int y, char symbol)
    {
        if (y >= 0 && y < gridSize)
        {
            polaGrid[y][x] = symbol;
        }
    }
};
```

```
void performOperation(const char *token, std::stack<double> &RPNstack)
{
    double operand2 = RPNstack.top();
    RPNstack.pop();
    double operand1 = RPNstack.top();
    RPNstack.pop();

    if (strcmp(token, "+") == 0)
    {
        RPNstack.push(operand1 + operand2);
    }
    else if (strcmp(token, "-") == 0)
    {
        RPNstack.push(operand1 - operand2);
    }
    else if (strcmp(token, "*") == 0)
    {
        RPNstack.push(operand1 * operand2);
    }
    else if (strcmp(token, "/") == 0)
    {
        RPNstack.push(operand1 / operand2);
    }
    else if (strcmp(token, "^") == 0)
    {
        RPNstack.push(pow(operand1, operand2));
    }
    else
    {
        RPNstack.push(0.0);
    }
}

};
```

1. **Pendekatan Solusi:** Kelas fungsipola: Representasi setiap fungsi dalam notasi Polandia Terbalik (RPN). Konstruktor menerima ekspresi RPN dan menyimpannya. Terdapat metode plotting yang digunakan untuk mengevaluasi fungsi pada setiap nilai  $x$  dari 0 hingga 20 dan menggambar plot fungsi dan turunannya pada matriks  $21 \times 21$ .
2. **Metode pola:** Metode ini menggunakan loop untuk iterasi melalui nilai  $x$  dari 0 hingga 20. Pada setiap iterasi, nilai fungsi dan turunan dihitung menggunakan metode `calcRPN` dan `calcTurunan`. Hasil perhitungan tersebut dibulatkan sesuai dengan aturan yang diberikan dan kemudian posisi yang sesuai dalam matriks  $21 \times 21$  ditandai dengan karakter '\*' untuk fungsi dan '+' untuk turunan.
3. **Metode cetak:** Metode ini menampilkan hasil pola pada grid. Melakukan looping dari baris paling bawah ke baris paling atas dan dari kolom kiri ke kolom kanan menggunakan `printf`.
4. **Metode nilaiBulat:** Metode ini digunakan untuk membulatkan nilai desimal menjadi bilangan bulat. Penggunaan metode ini terdapat pada metode pola dalam kelas fungsipola. Metode ini memainkan peran penting dalam mengubah nilai yang diperoleh dari perhitungan fungsi atau turunan menjadi nilai yang sesuai dengan aturan penulisan pada matriks  $21 \times 21$ .
5. **Metode calcRPN:** Metode ini mengimplementasikan evaluasi ekspresi RPN menggunakan stack. Setiap token dalam ekspresi diproses, dan hasilnya dihitung dengan memanipulasi stack sesuai dengan operator dan operand yang ditemukan.
6. **Metode calcTurunan:** Metode ini menghitung turunan numerik dengan menggunakan rumus beda hingga. Nilai fungsi dievaluasi pada titik  $x+h$  dan  $x-h$ , kemudian turunan dihitung sebagai perbedaan nilai fungsi dibagi dengan dua kali nilai  $h$ .
7. **Metode setGridPoint:** Metode ini memastikan bahwa nilai  $y$  yang akan ditandai pada matriks berada dalam rentang yang valid (antara 0 dan `gridSize`). Ini membantu menghindari kesalahan saat menetapkan karakter pada matriks.
8. **Metode performOperation:** Metode ini menangani operasi yang ditemukan dalam ekspresi RPN. Operand-operand diambil dari stack, operasi dilakukan, dan hasilnya kembali dimasukkan ke dalam stack.

#### 4. Main

```
int main()
{
    int functions;
    scanf("%d", &functions);

    int c;
    while ((c = getchar()) != '\n' && c != EOF)
        ;

    char grid[gridSize][gridSize];

    for (int i = 0; i < functions; ++i)
    {
        char expression[maxExpressionSize];
        scanf("%999[^\n]", expression);
        getchar();

        fungsipola plotter(expression);

        for (int y = 0; y < gridSize; ++y)
        {
            for (int x = 0; x < gridSize; ++x)
            {
                grid[y][x] = '.';
            }
        }
        plotter.pola(grid);
        plotter.cetak(grid);
    }

    return 0;
}
```











### **Pendekatan Fungsi Main:**

1. **Membaca Jumlah Fungsi:** Digunakan `scanf` untuk membaca jumlah fungsi dari input. Jumlah fungsi ini penting sebagai batas iterasi untuk membaca dan memproses setiap ekspresi fungsi.
2. **Membersihkan Buffer:** Setelah membaca jumlah fungsi, dilakukan penggunaan `while` loop untuk membersihkan buffer. Ini diperlukan agar buffer tidak berisi karakter `newline` atau karakter lain yang dapat memengaruhi proses membaca ekspresi fungsi selanjutnya menggunakan `scanf` atau `getline`.
3. **Membaca dan Memproses Setiap Ekspresi Fungsi:** Digunakan `loop for` untuk melakukan iterasi sebanyak jumlah fungsi. Ekspresi fungsi dibaca menggunakan `scanf` untuk membaca karakter sampai karakter `newline`. Kemudian, `getchar()` digunakan untuk mengonsumsi karakter `newline` yang masih tersisa di buffer setelah `scanf`. Ini diperlukan agar tidak ada karakter `newline` yang ikut dibaca saat membaca ekspresi fungsi selanjutnya.
4. **Membuat dan Menampilkan Plot:** Menggunakan matriks `grid` untuk menyimpan hasil plot dari setiap fungsi. Objek `functipola` dibuat untuk setiap ekspresi fungsi dan diinisialisasi dengan ekspresi tersebut. Matriks `grid` diinisialisasi dengan karakter `'.'` untuk setiap elemennya. Metode `pola` dari objek `functipola` dipanggil untuk membuat plot fungsi dan turunannya pada matriks `grid`. Hasil plot ditampilkan dengan looping dari baris paling bawah ke baris paling atas dan dari kolom kiri ke kolom kanan menggunakan `printf`.
5. **Mengakhiri Program:** Program diakhiri dengan mengembalikan nilai 0 dari fungsi `main` sebagai indikasi bahwa program telah berakhir dengan sukses.

Dengan menjalankan kode tersebut, bisa dilihat hasil seperti dibawah ini bahwa kode tersebut bisa menyelesaikan problem tersebut

Dan dibawah bukti Verdict Accept dengan beberapa percobaan untuk mencapai waktu 0.0ms dengan memory 5.1MiB, namun setelah dilakukan berulang kali, hanya didapatkan 0.0 dengan memory 5.2MiB atau 0.1 dengan memory 5.1MiB, hal ini dilakukan setelah sebelumnya melakukan beberapa peningkatan, dimana awalnya kode memakai terlalu banyak library dan mengganti beberapa library yang terlalu sedikit digunakan.

## : submissions

ID		DATE	PROBLEM	RESULT	TIME	MEM	LANG
32184711		2023-11-16 16:25:44	Plotting functions (variation)	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.00	5.2M	CPP14
32184709		2023-11-16 16:25:32	Plotting functions (variation)	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.01	5.2M	CPP14
32178785		2023-11-15 15:20:59	Plotting functions (variation)	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.01	5.1M	CPP14
32178781		2023-11-15 15:20:45	Plotting functions (variation)	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.01	5.2M	CPP14
32176385		2023-11-15 06:13:19	Plotting functions (variation)	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.01	5.1M	CPP14
32176284		2023-11-15 05:49:06	Plotting functions (variation)	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.01	5.1M	CPP14
32169273		2023-11-13 17:14:07	Plotting functions (variation)	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.00	5.2M	CPP14
32167809		2023-11-13 12:18:05	Plotting functions (variation)	<b>accepted</b> <a href="#">edit</a> <a href="#">ideone.it</a>	0.01	5.2M	CPP14

Terima Kasih telah membaca, Waalaikumsalam wr wb.