Vizualizace grafu matematické funkce

KIV/PC – Semestrální práce

Student: Ladislav Čákora
Osobní číslo: A23B0149P
Email: cakoral@students.zcu.cz

Obsah

1 Úvod

Toto je dokumentace k aplikaci pro*vizualizaci matematické funkce v souboru postscript*, která byla vytvořena jako semestrální práce z předmětu**Programování v jazyce C**. Celá aplikace je vytvořena v jazyce C podle standardu ANSI.

2 Zadání práce

Naprogramujte v **ANSI C** přenositelnou konzolovou aplikaci, která jako vstup načte z parametrů na příkazové řádce matematickou funkci ve tvaru $y = f(x); x \in \mathbf{R}$, provede její analýzu a vytvoří soubor ve formátu PostScript s grafem této funkce na zvoleném definičním oboru. Program se bude spouštět příkazem:

```
graph.exe {func} {out-file} [limits]
```

Symbol `func` zastupuje zápis matematické funkce jedné reálné nezávislé proměnné (funkce ve více dimenzích program řešit nebude), nalezené-li během analýzy zápisu funkce více než jednu nezávislou proměnnou, vypíše srozumitelné chybové hlášení a skončí. Závislá proměnná (zde y) je implicitní, a proto se její zápis na příkazové řádce nebude uvádět. Symbol `out-file` zastupuje jméno výstupního PostScriptového souboru. Takže váš program může být během testování ověřen spuštěním například takto:

```
X:\\>graph.exe "sin(2x)*3" mygraph.ps
```

Výsledkem práce programu bude soubor ve formátu PostScript, který bude zobrazovat graf zadané matematické funkce – ve výše uvedeném případě $y = \sin(2x) \cdot 3$ – v kartézské soustavě souřadnic $O : (x, y)$ s vyznačenými souřadnými osami a (alespoň) vyznačenými hodnotami definičního oboru a oboru hodnot funkce (viz Specifikace výstupu programu).

Pokud nebudou na příkazové řádce uvedeny alespoň dva argumenty, vypište chybové hlášení a stručný návod k použití programu v angličtině podle běžných zvyklostí (viz např. ukázková semestrální práce na webu předmětu Programování v jazyce C). **Vstupem programu jsou pouze argumenty na příkazové řádce – interakce s uživatelem pomocí klávesnice či myši v průběhu práce programu se neočekává.**

3 Analýza úlohy

K vyřešení úlohy jsou zapotřebí tři hlavní kroky:

1. Zpracování vstupu do podoby, se kterou je snazší pracovat
2. Převod výrazu z infixové notace do postfixového zápisu a jeho následné vyhodnocení
3. Vykreslení grafu, resp. tvorba postscriptových příkazů

3.1 Zpracování vstupu

Je třeba zpracovat dva vstupy: funkce a limity pro vykreslení. Existenci limit můžeme zjistit spočtením dvojteček v posledním parametru. Pokud jsou tři, můžeme zkontrolovat čísla. První tři musí být následována dvojtečkou, čtvrté ukončovacím znakem řetězce v jazyce C. Pokud nenajdeme žádnou dvojetčku, limity nebyly zadány. Pokud najdeme jiný počet, je vstup neplatný, jelikož v žádném z předchozích parametrů se vyskytovat nesmí.

Parametr výstupního souboru se nijak zpracovávat nemusí, pouze se uloží pro pozdější použití.

Zpracování funkce je složitější, jelikož se v ní může vyskytovat několik druhů znaků a zároveň nemusí být pouze v jednom parametru. Kvůli tomu se musí odpovídající parametry pospojovat, je také vhodné odstranit mezery a přepsat desetinné čárky na tečky.

K řešení mi přijde nejvhodnější využít metodu rozdělení funkce na tokeny, jelikož mě dále obvykle bude zajímat hlavně typ tokenu, spíše než samotná hodnota, kterou převážně potřebuji až u samotného výpočtu.

Další možností by bylo převést výraz na stromovou strukturu pomocí rekurzivního sestupu, ale to mi přijde vhodnější spíše pro implementaci v nějakém objektově orientovaném jazyce, v C se mi zdá vhodnější zvolená implementace.

3.2 Vyhodnocení výrazu

Tento krok se skládá ze tří částí. Nejprve je nutné zkontrolovat matematickou správnost zápisu, především správné spárování závorek a vhodnou posloupnost tokenů. Poté je třeba převést výraz z infixové notace do postfixové, která je značně jednodušší na vyhodnocení. K tomu lze využít algoritmus **Shunting-yard** s lineární složitostí. Dalším krokem je vyhodnocení infixového výrazu. To bude nutné provést pro vhodné husté, rovnoměrné pokrytí definičího oboru daného uživatelem.

3.3 Vykreslení grafu

K vykreslení v jazyce **PostScript** bude třeba provést transformaci vypočítaných hodnot pro pokrytí grafického prostoru grafu definičním oborem a oborem hodnot zadaných uživatelem. Následně příslušné příkazy zapíšeme do výstupního souboru.

4 Implementace

4.1 Modul main.c

4.1.1 Předdefinovaná makra a proměnné

Tabulka 1: Tabulka definovaných konstant

Makro	**Hodnota**	**Význam**
STEPS	2000	Počet hodnot
EXIT_INVALID_ARGS	1	Nesprávně zadané argumenty
EXIT_INVALID_FUNCTION	2	Nesprávně zadaná funkce
EXIT_INVALID_OUTPUT	3	Nesprávný výstupní soubor
EXIT_INVALID_LIMITS	4	Nesprávně zadané limity

4.1.2 int check_if_correct(Expression *expr)

Funkce ověří správnost zadaného výrazu podle pravidel matematického zápisu:

- **nesmí začínat** binárním operátorem nebo pravou závorkou
- po levé závorce **nesmí** být binární operátor nebo pravá závorka
- po pravé závorce **musí** být binární operátor, pokud už není na konci
- po binárním nebo unárním operátoru **nesmí** být binární operátor nebo pravá závorka
- po funkci **musí** být levá závorka
- po čísle, proměnné nebo pravé závorce **musí** být binární operátor nebo pravá závorka
- **nesmí končit** binárním operátorem, levou závorkou nebo funkcí

Zároveň všechny závorky musí být spárované, což se ověří pomocí zásobníku - *levá –> push, pravá –> pop.* Funkce vrací **1** pokud výraz prošel, **0** pokud ne.

4.1.3 int get_limits(char *arg, double *x_min, double *x_max, double *y_min, double *y_max)

Zkontroluje správnost zápisu limit a uloží je na příslušné adresy předané jako argumenty. Při neúspěchu vrací **0**, při úspěchu **1**.

4.1.4 int main(int argc, char *argv[])

Vstupní bod programu. Má na starosti hlavní běh programu a volání jednotlivých částí řešení.

Může vracet takové hodnoty

- **0** - Vše proběhlo v pořádku
- **1** - Nebyly předány správné parametry na příkazové řádce při spuštění programu.
- **2** - Řetězec předaný programu jako první parametr není akceptovatelným zápisem matematické funkce (neobsahuje právě jednu proměnnou x, obsahuje nepovolené operátory, symboly, atp.).
- **3** - Řetězec předaný programu jako druhý parametr nemůže být v hostitelském operačním systému názvem souboru nebo takový soubor nelze vytvořit či do něj ukládat informace.
- **4** - Uvedený nepovinný třetí parametr nelze dekódovat jako rozsahy zobrazení, protože např. obsahuje nepovolené znaky nebo uvedené hodnoty nedávají smysl, apod.

4.2 Modul stack.h

4.2.1 struct stack

Struktura reprezentující zásobník ukazatelů na tokeny.

- **Token **items** - ukazatel na pole ukazatelů na token, položky v zásobníku
- **int sp - stack pointer**, index vrcholu zásobníku
- **unsigned int size** - velikost zásobníku

4.2.2 stack *stack_create(unsigned int size)

Funkce k vytvoření zásobníku tokenů o velikosti `size`. Vrací **ukazatel na zásobník** nebo **NULL** při neúspěchu.

4.2.3 int stack_push(stack *s, Token *item)

Vloží `item` na zásobník. Vrátí **0** při neúspěchu, jinak **1**.

4.2.4 Token *stack_pop(stack *s)

Vrátí **token z vrcholu** zásobníku a **odebere** ho, pokud byl prázdný, vrátí **NULL**.

4.2.5 Token *stack_peek(stack *s)

Vrátí **token z vrcholu** zásobníku ale **neodebere** ho, pokud byl prázdný, vrátí **NULL**.

4.2.6 void stack_clear(stack **s)

Vyprázdní zásobník a uvolní jemu alokovanou paměť

4.3 Modul tokens.h

4.3.1 Předdefinovaná makra a konstanty

Tabulka 2: Tabulka definovaných konstant

Makro	**Hodnota**	**Význam**
VAR	'x'	Znak proměnné
OPS	"+-*/^"	Znaky operátorů
FUNCTIONS	{"asin", "acos", ..., "ln"}	Řetězce pro volání funkcí

4.3.2 enum TokenType

Výčet možných typů tokenů

- TOKEN_NUMBER
- TOKEN_VARIABLE
- TOKEN_BINARY_OPERATOR
- TOKEN_LPARENTHESIS
- TOKEN_RPARENTHESIS
- TOKEN_FUNCTION
- TOKEN_UNARY_OPERATOR
- TOKEN_UNKNOWN

4.3.3 struct Token

Struktura reprezentující token.

- **TokenType type** - typ tokenu
- **char *value** - řetezec hodnoty tokenu

4.3.4 struct Expression

Struktura reprezentující výraz jako pole ukazatelů na Token.

- **Token **arr** - pole ukazatelů
- **int len** - délka pole

4.3.5 Expression *tokenize(char **expr)

Převede výraz reprezentovaný polem řetězců na tokeny. Nejprve si přpraví řetězec funkcí `remove_spaces_replace_commas (4.3.6)`, následně tento řetězec projde a rozpozná token příslušející danému znaku. Vrátí **NULL**, pokud se nepodařilo převést řetězec na tokeny, jinak vrátí **výraz jako ukazatel na Expression**.

4.3.6 int remove_spaces_replace_commas(char *str)

Odstraní mezery v řetězci `str` a nahradí desetinné tečky za čárky. Vrátí **délku upraveného řetězce** při úspěchu, **0** jinak.

4.3.7 void free_expression(Expression **expr)

Uvolní paměť alokovanou pro jednotlivé tokeny, pole ukazatelů na ně a nakonec i samotnou strukturu Expression.

4.3.8 Pomocné funkce

Modul dále obshuje funkce sloužící k rozpoznávání jednotlivých typů tokenů. Ty ale nejsou obsažené v hlavičkovém souboru, jelikož by nemusely v jiném kontextu fungovat tak jak bylo zamýšleno. Funkce pro rozpoznání proměnných, operátorů a funkcí jsou jednoduché, vycházejí pouze z porovnávání se vzorem. Funkce pro rozpoznání unárního mínus ke správné funkci potřebuje i předchozí znak, aby se mohla správně rozhodnout.

Za zmínku stojí hlavně funkce pro rozpoznání čísel `is_number`. Ta rozpoznává čísla ve formátu `d.dEsd` kde **d** je řetězec dekadických číslic, **s** je znaménko plus nebo mínus a **E**, případně **e** je značka pro začátek exponentu. Důležité je, aby před E byla alespoň jedna číslice. Pokud máme celé číslo, tečka samozřejmě není povinná, stejně jako exponent. Pokud se ale E v řetězci objeví, musí být následováno alespoň jednou číslicí.

4.4 Modul yard.h

4.4.1 int shunting_yard(Expression *exp)

Převede výraz v infixové notaci v `*token_arr` do postfixové notace pomocí algoritmu Shunting-yard a uloží přepsaný výraz zpět do `exp` . Vrátí **0**, pokud se vyskytla chyba, jinak vrací **nový počet tokenů**. Ten se mohl změnit, protože algoritmus odstraňuje závorky. Zároveň upraví hodnotu i pro `exp->len`.

4.4.2 Algoritmus Shunting-yard

Algoritmus **Shunting-yard** slouží k převodu matematických výrazů z infixové notace (např. a+b) do postfixové (RPN, např. ab+).

Vstup zpracovává sekvenčně:

- Čísla a proměnné přidává do výstupní fronty.
- Operátory ukládá na zásobník dle priority.
- Levá závorka se vkládá na zásobník.
- Pravá přesouvá operátory ze zásobníku do výstupu až k levé závorce.

Po dokončení vstupu se zásobník vyprázdní do výstupní fronty.

4.4.3 int priority(Token *t)

Vrátí **prioritu tokenu** pro algoritmus Shunting-yard. Pokud token nerozpozná, vrátí **0**.

4.5 Modul postfix_eval.h

4.5.1 Předdefinovaná makra a konstanty

Tabulka 3: Tabulka definovaných konstant

Makro	**Hodnota**	**Význam**
NOTANUMBER	1 000 001	Funkce nebyla v daném bodě vyhodnotitelná
MAX	1 000 000	Horní hranice hodnot
MIN	-1 000 000	Spodní hranice hodnot

4.5.2 enum functions

Výčet funkcí

- F_UNDEFINED
- SIN
- COS
- TAN
- SINH
- COSH
- TANH
- ASIN
- ACOS
- ATAN
- ABS
- LOG
- LN
- EXP

4.5.3 enum operators

Výčet operátorů

- O_UNDEFINED
- ADD
- SUB
- MUL
- DIV
- POW

4.5.4 int postfix_eval(Token *token_arr, int tokens_count, double x, double *result)

Vyhodnotí postfixový výraz z `token_arr` s parametrem `x`. K vyhodnocení se vytvoří kopie clého pole včetně jednotlivých tokenů, na které se pole odkazuje. Výsledek se uloží do `*result` . Vrátí **0** v případě chyby, jinak **1**.

4.5.5 Vyhodnocení postfixového výrazu

Prochází se výraz token po tokenu:

- čísla se vloží do zásobníku
- pro operátory a funkce se ze zásobníku vezme příslušný počet parametrů, vyhodnotí se a výsledek se vrátí do zásobníku

Výsledek výrazu zůstane jako poslední v zásobníku.

4.5.6 Pomocné funkce

Modul dále obsahuje pomocné funkce pro vyhodnocování operátorů a funkcí. Funkce by nemusely poskytovat zamýšlenou funkcionalitu, pokud by byly použity v jiném kontextu, proto nejsou uvedeny v hlavičkovém souboru. Funkce `get_operator` a `get_function` slouží ke zjištění, o který operátor nebo funkci se vlastně jedná. Toho využívají funkce `eval_function` a `eval_operator` , které vyhodnotí funkci nebo operátor pro zadané parametry.

4.6 Modul postscript.h

4.6.1 Předdefinovaná makra a konstanty

Tabulka 4: Tabulka definovaných konstant

Makro	**Hodnota**	**Význam**
GRAPH_WIDTH	1400	Šířka grafu v pixelech
GRAPH_HEIGHT	1000	Výška grafu v pixelech
PAGE_WIDTH	1600	Šířka stránky v pixelech
PAGE_HEIGHT	1200	Výška stránky v pixelech
OFFSET	100	Okraj kolem grafu

4.6.2 double transform_x(double x, double x_min, double x_max)

Upraví hodnoty na vodorovné ose podle vztahu:

$$\text{transform}_x(x, x_{\min}, x_{\max}) = (x - x_{\min}) \cdot \frac{\text{GRAPH\_WIDTH}}{x_{\max} - x_{\min}} + \text{OFFSET}$$

4.6.3 double transform_y(double y, double y_min, double y_max)

Upraví hodnoty na svislé ose podle vztahu:

$$\text{transform}_y(y, y_{\min}, y_{\max}) = (y - y_{\min}) \cdot \frac{\text{GRAPH\_HEIGHT}}{y_{\max} - y_{\min}} + \text{OFFSET}$$

4.6.4 int create_postscript(double *x, double *y, int length, double x_min, double x_max, double y_min, double y_max, char *outpath)

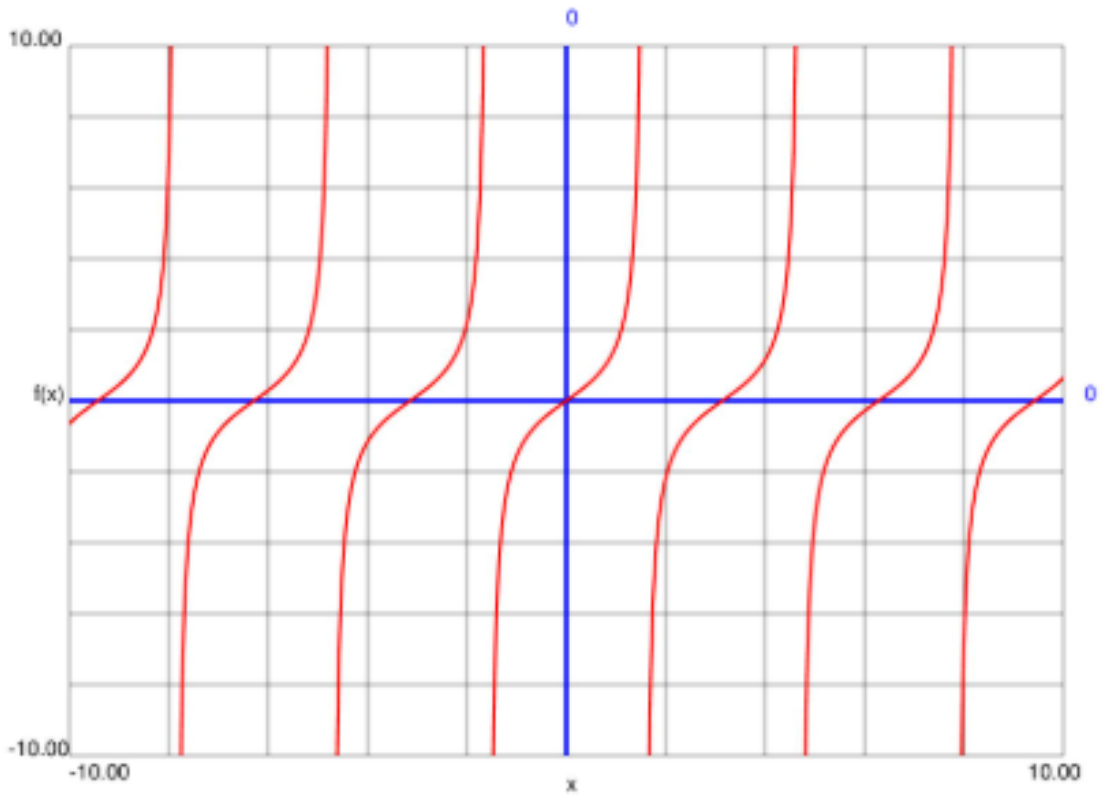
Vykreslí graf do souboru postscript dle zadaných parametrů. Vrací **1**, pokud nenastala chyba, jinak vrací **0**. Během vykreslování je nutné dopočítat okrajové hodnoty, kde počáteční nebo koncový bod leží mimo vykreslovanou oblast. Toho lze dosáhnout zjištěním předpisu přímky mezi těmito body, následným nalezení inverzní funkce a dosazením horního, případně spodního okraje. Směrnici přímky najdeme jako podíl vzdálenosti po svislé a po vodorovné vzdálenosti, posun zjistíme dosazením jednoho z bodů a směrnice. Inverzní funkci najdeme vyjádřením x z nalezeného předpisu funkce. Celý vztah potom vypadá takto:

$$x = \frac{y - y_1 + \frac{y_2 - y_1}{x_2 - x_1} \cdot x_1}{\frac{y_2 - y_1}{x_2 - x_1}}$$

Kde bod (x, y) je hledaný bod na okraji grafu, bod (x_1, y_1) je počáteční bod úsečky.

Bod (x_2, y_2) je koncový bod úsečky.

Výsledný graf může vypadat třeba takto:



Obrázek 1: Graf funkce `tan(x)`

5 Uživatelská příručka

5.1 Přeložení a sestavení programu

Pro sestavení programu pomocí make je třeba mít nainstalovaný překladač gcc. Příkaz pro sestavení:

```
make clean
make
```

Pokud se nezdaří z důvodu absence nástroje **make**, což se nejčastěji stane na systémech Windows, je možné vyzkoušet příkazy poskytované nástrojem **mingw-32**, jehož součástí je případně i překladač gcc:

```
mingw-32 make clean
mingw-32 make
```

5.2 Spuštění

Příkaz pro spuštění:

```
graph.exe <funkce> <výstup> [<x_min:x_max:y_min:y_max>]
```

Parametr limitů v hranatých závorkách je nepovinný. Spuštění příkazem:

```
graph.exe "tan(x)" out.ps
```

Vytvoří výše ukázaný graf (tady 4.6.4)

5.3 Parametry

5.3.1 Funkce

Matematická funkce. Povolené operátory jsou: *+, -, *, / ,* ^. Povolené funkce jsou: ***sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, abs, exp, log, ln***. Výraz dále může obsahovat reálná čísla (formát zde 4.3.8), proměnnou x a kulaté závorky. Výraz může být zadaný s mezerami, pokud bude uzavřen do uvozovek. To se doporučuje i u zápisu bez mezer, protože ne každý terminál přijme všechny znaky jako text (například ^v CMD na Windows). Mohlo by poté dojít k převodu na jinou, nezamýšlenou funkci (třeba znovu CMD: parametr 2^32 načte jako 232).

5.3.2 Výstup

Název ýstupního soubor by měl končit příponou .ps a splňovat požadavky Vašeho opečťačního systému.

5.3.3 Limity

Limity jsou nepovinný parametr, při jejich neuvedení se budou předpokládat limity -10 až 10 na obou osách. Mohou to být reálná čísla oddělená dvojtečkami.

6 Závěr

Výstupem semestrální práce je konzolová aplikace na převod matematické funkce na graf souboru PostScript. Aplikace by se v budoucnu dala rozšířit o další funkce a operátory. Zároveň by se dala vylepšit grafická stránka výstupu.