ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Předmět KIV/PC
semestrální práce na téma:

VIZUALIZACE GRAFU MATEMATICKÉ FUNKCE

Autor: Jakub Štrunc
Datum: 30. prosince 2024

Obsah

1 Úvod **4**

2 Kompletní zadání **5**

3 Analýza úlohy **8**
3.1 Kontrola vstupu 8
3.2 Rozdělení funkce 9
3.3 Kontrola zápisu fukce 9
3.4 Kontrola limit 9
3.5 Vytvoření PostScript souboru 9
3.6 Převod výrazu do polské reverzní notace 10
3.7 Kontrola vytvoření souboru 12
3.8 Vykresleni grafu 12
3.9 Vyhodnocení výrazu v polské reverzní notaci 12

4 Popis implementace **14**
4.1 Hlavní program 14
4.1.1 Funkce `add_spaces` 14
4.1.2 Funkce `is_valid_function` 15
4.1.3 Hlavní funkce `main` 16
4.2 PostScript dokument 18
4.2.1 Funkce `create_postscript` 18
4.2.2 Funkce `draw_square_axis` 19
4.2.3 Funkce `draw_ticks_and_labels` 19
4.2.4 Funkce `draw_graph` 19
4.2.5 Funkce `close_postscript` 20
4.3 Vyhodnocení matematických výrazů 20
4.3.1 Funkce `evaluate_function` 20
4.3.2 Funkce `evaluate_postfix_expression` 21
4.4 Shunting yard algoritmus 23
4.4.1 Funkce `strdup` 23
4.4.2 Funkce `precedence` 24
4.4.3 Funkce `is_left_associative` 24
4.4.4 Funkce `is_operator` 24
4.4.5 Funkce `shunting_yard` 24
4.4.6 Funkce `is_function` 26
4.5 Fronta 26
4.6 Zásobník 27

5 Uživatelská příručka **28**
5.1 Postup přeložení programu 28
5.1.1 Linux 28
5.1.2 Windows 28
5.2 Spuštění programu 28
5.3 Výstupy programu 31
5.4 Ukázky běhu programu pro různá vstupní data 32

6 Závěr **34**

Kapitola 1

Úvod

Tato semestrální práce se zaměřuje na implementaci programu pro vizualizaci matematických funkcí. Cílem práce je vytvořit přenosnou konzolovou aplikaci v jazyce C, která bude schopna přijímat matematické funkce na příkazové řádce a generovat grafy funkcí ve formátu PostScript. Program bude zpracovávat jednorozměrné matematické funkce a umožní definování rozsahů zobrazení grafu.

Práce se zaměřuje na akceptování základních matematických funkcí, jako jsou absolutní hodnota, logaritmy, goniometrické a cyklometrické funkce atd. Funkce bude možné zadat s využitím standardních operátorů, jako jsou sčítání, odčítání, násobení a dělení, a s podporou závorek pro definici precedence operací. Program bude rovněž umožňovat zadání rozsahu pro definiční obor a obor hodnot, a to jak pro grafy na intervalu, tak i pro automatické nastavení implicitního rozsahu.

Výsledkem práce bude aplikace, která umožní generování vizualizovaných grafů matematických funkcí.

Kapitola 2

Kompletní zadání

Zadání

Naprogramujte v ANSI C přenositelnou konzolovou aplikaci, která jako vstup načte z parametru na příkazové řádce matematickou funkci ve tvaru $y = f(x)$, kde $x, y \in \mathbb{R}$, 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). Pokud během analýzy zápisu funkce nalezne více nezávislých proměnných než jednu, vypíše srozumitelné chybové hlášení a skončí. Závislá proměnná y je implicitní, a proto se levá strana rovnosti v zápisu nebude uvádět. Symbol $\langle out - file \rangle$ zastupuje jméno výstupního PostScript souboru.

Program může být během testování spuštěn například takto:

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

Výsledkem práce programu bude soubor ve formátu PostScript, který bude zobrazovat graf zadané matematické funkce – ve výše uvedeném případě $y = \sin(2x)^3$ – v kartézské souřadnicové soustavě $(O; x, y)$ s vyznačenými souřadnými osami a (aspoň) význačnými hodnotami definičního oboru a oboru hodnot funkce.

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**.

Specifikace vstupu programu

Vstupem programu jsou pouze parametry na příkazové řádce. Prvním (a nejdůležitějším) parametrem je matematická funkce, kterou má program zpracovat. S ohledem na to, že její zápis může obsahovat mezery, musí program akceptovat jak zápis funkce obklopený uvozovkami, tak bez nich. To znamená, že program musí správně fungo-

vat při spuštění s příkazem `graph.exe` $x+x$`^2` i `graph.exe "`$x \ + \ x$ `^ 2"`. Použití mezer v zápisu funkce je libovolné. Pokud zápis parametru není uzavřen v uvozovkách, nesmí obsahovat mezery, jinak by byl příkazovým procesorem ve Windows vyhodnocen jako několik parametrů. Naopak pokud je třeba do zápisu funkce vložit mezery (z estetických důvodů nebo kvůli přehlednosti), musí být zápis uzavřen v uvozovkách. Umožněte proto oba způsoby zápisu, tedy „hustý“ zápis bez mezer i zápis s mezerami uzavřený do uvozovek. Je také možné, že uživatel použije mezery různě v zápisu, například takto: "$sin \ \ (x$ `^2`$) * cos(x \ / \ 2.0)$".

Způsob zápisu funkcí

V zápisu matematické funkce, jejíž graf bude program generovat, mohou být použity následující symboly: (i) konstanty, (ii) proměnná, (iii) aritmetické operátory, (iv) funkce, (v) závorky. Jiné symboly v zápisu nejsou povoleny – pokud se v zápisu objeví, program by měl reagovat srozumitelným chybovým hlášením.

Konstanty (i) mohou být uvedeny ve všech formátech, které jsou v jazyce C akceptovatelné pro zápis celého nebo reálného čísla, např. $.5E-02$ nebo $1E0$. Vzhledem k tomu, že program má za úkol generovat pouze 2D grafy, jedinou nezávislou proměnnou (ii), která se může objevit v zápisu funkce, je x. Tato proměnná není nutné deklarovat, protože je implicitně přítomná a připravena k použití uživatelem.

Tabulka níže uvádí přehled aritmetických operátorů (iii), které mohou být použity při zápisu matematické funkce:

Operace	Zápis operátoru	Příklad
Unární mínus	-	-x
Sčítání	+	x + 2
Odčítání	-	x - 2
Násobení	*	2 * x
Dělení	/	x / 2
Umocnění	^	x^2

Jiné operátory nejsou povolené. Priorita operátorů je stanovena matematickými pravidly a může být upravena závorkami (a) – jiné druhy závorek v zápisu funkce nesmí být použity. Platný zápis je například: (x * sin(x^(2^x))) / 2.

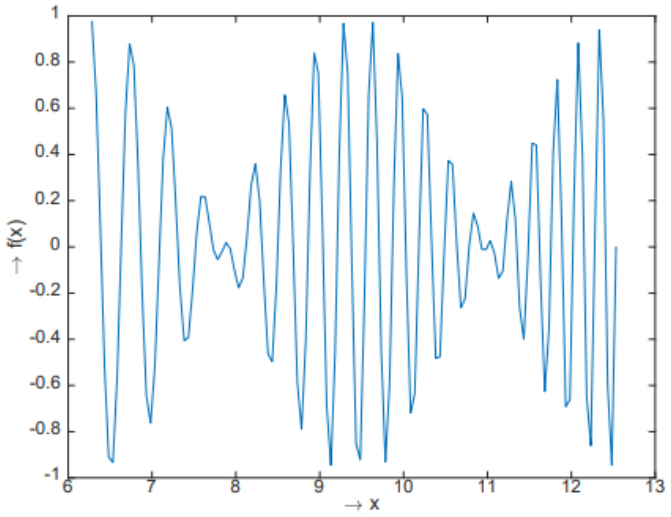
Specifikace rozsahu zobrazení

Třetí – **nepovinný** – parametr $[<limits>]$ na příkazové řádce slouží k předání informací o rozsazích zobrazení grafu analyzované funkce, tedy o horní a dolní mezi definičního oboru a oboru hodnot funkce. Má tvar $\langle x_{\text{dol}}\rangle : \langle x_{\text{hor}}\rangle : \langle y_{\text{dol}}\rangle : \langle y_{\text{hor}}\rangle$. Chceme-li tedy například zobrazit funkci $y = \sin(x^2)\cdot\cos(x)$ na intervalu $\langle 2\pi; 4\pi\rangle$, přičemž obor hodnot funkce bude roztažen přes celý graf, spustíme program příkazem: `graph.exe "`$sin(x^2) * cos(x)$`" sin2cos.ps 6.28:12.56:-1:1`. Obrázek 2.1 ukazuje, jak by měl vypadat výsledek.

Není-li rozsah uveden (protože se jedná o nepovinný parametr), použije se implicitní nastavení, které je pro definiční obor i obor hodnot $\langle -10; 10\rangle$.

Matematické funkce v zápisu

Program by měl akceptovat v zápisu zobrazované funkce některé běžně používané matematické funkce. Funkce uvedené v následujícím výčtu program **musí** akceptovat, jinak nebude uznán za řádně dokončený: absolutní hodnota `abs`, funkce `exp`, přirozený logaritmus ln, dekadický logaritmus log; goniometrické funkce sin, cos, tan; cyklometrické funkce arcsin, arccos, arctan; hyperbolometrické funkce sinh, cosh, tanh. Další funkce můžete samozřejmě implementovat z vlastní iniciativy.

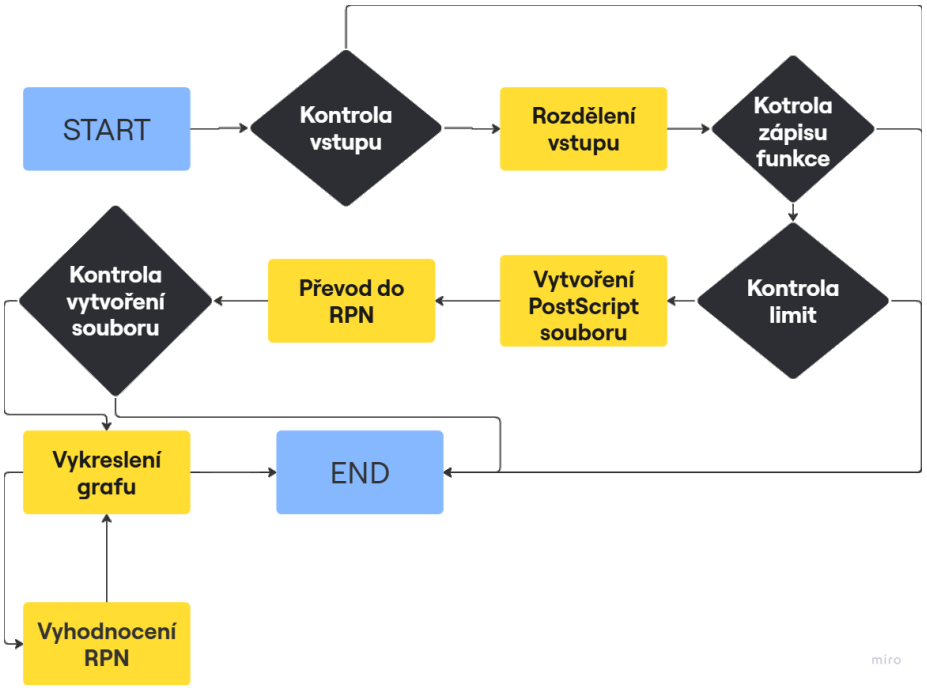


Obrázek 2.1: Graf funkce $y = sin(x^2) \cdot cos(x)$ na intervalu $\langle 2\pi; 4\pi \rangle$.

Kapitola 3

Analýza úlohy

K naprogramování konzolové aplikace pro zpracování matematických funkcí do Postscriptu je nutné tento proces rozdělit na několik klíčových fází, které jsou znázorněny na přiloženém diagramu.



Obrázek 3.1: Diagram procesu aplikace.

3.1 Kontrola vstupu

Prvním krokem v procesu je kontrola vstupních parametrů, která ověřuje, zda uživatel zadal správný počet argumentů. Pokud argumenty neobsahují exe soubor, funkci a výstupní PostScript dokument, program vyhodí chybovou hlášku a ukončí se.

3.2 Rozdělení funkce

Po validaci vstupu je dalším zásadním krokem rozdělení matematické funkce na jednotlivé operátory a operandy. Tento krok je nezbytný pro následné zpracování funkce, protože nám umožňuje jednodušší čtení operátorů a operandů v dalších krocích programu, čímž se zajišťuje efektivní práce s výrazy.

Problémy, které tento krok přináší:

1. **Správné rozdělení operátorů a operandů**: Je nezbytné správně rozlišovat mezi operátory, operandy a funkcemi a přidat mezery tam, kde je to potřeba, aniž by došlo k narušení struktury výrazu.
2. **Přidání mezer kolem operátorů**: Program musí přidat mezery před a za každým operátorem (např. +, -, *, /, ^), ale zároveň zajistit, aby mezery nezasahovaly do samotného zápisu funkcí nebo závorek.
3. **Rozpoznání funkcí**: Program musí správně identifikovat funkce a přidat mezery mezi funkcí a otevírací závorkou, pokud tam mezera chybí.

Řešení:

Program analyzuje řetězec, postupně porovnává každý znak s definovanými vzory a přidává mezery mezi operátory, funkcemi a závorkami. Operátory a závorky jsou doplněny mezerami, pokud již nejsou přítomné.

3.3 Kontrola zápisu fukce

Zkontroluje, zda řetězec předaný programu jako první parametr je akceptovatelným zápisem matematické funkce, tj. zda obsahuje právě jednu nezávislou proměnnou x, a neobsahuje nepovolené operátory, symboly nebo jinou neplatnou syntaxi. Pokud není tento zápis platný, program vyhodí chybovou hlášku a ukončí se.

3.4 Kontrola limit

Zkontroluje, zda řetězec předaný programu jako vstupní parametr obsahuje limity ve správném formátu $\langle x_{\text{dol}} \rangle : \langle x_{\text{hor}} \rangle : \langle y_{\text{dol}} \rangle : \langle y_{\text{hor}} \rangle$. Dále ověřuje, zda limity dávají smysl, tj. zda minimální hodnoty nejsou větší nebo rovny maximálním hodnotám. Pokud nejsou tyto podmínky splněny, program vyhodí chybovou hlášku a ukončí se.

3.5 Vytvoření PostScript souboru

Po zkontrolování vstupních údajů je dalším zásadním krokem vytvoření PostScript struktury, která slouží k přípravě dat pro generování PostScript souboru. Soubor musíme nejprve vytvořit a otevřít pro zápis. Následně se připraví základní parametry, jako jsou rozsahy funkce, a zapíše se hlavička dokumentu.

3.6 Převod výrazu do polské reverzní notace

Pro budoucí zpracování funkce je klíčovým krokem převod do polské reverzní notace (RPN). Tento převod je nezbytný, protože zajišťuje jednoznačné vyhodnocení matematického výrazu bez potřeby závorek a složitých pravidel priorit operátorů.

Problémy, které tento krok přináší:

1. **Rozpoznání tokenů**: Program musí správně identifikovat různé typy tokenů, jako jsou čísla, operátory, proměnné, funkce a závorky.
2. **Respektování priority operátorů**: Je důležité správně zohlednit prioritu operátorů (např. násobení má vyšší prioritu než sčítání).
3. **Zpracování funkcí**: Funkce musí být správně identifikovány a převedeny do formátu, který umožní jejich správné vyhodnocení v polské reverzní notaci.
4. **Manipulace se závorkami**:Správné párování a odstranění závorek je zásadní pro zachování správné struktury výrazu.

Řešení:

Převod výrazu do polské reverzní notace (RPN) je prováděn pomocí algoritmu Shunting Yard. Tento algoritmus zpracovává jednotlivé tokeny výrazu a rozhoduje, zda je zařadí do výstupní fronty, nebo uloží do zásobníku podle jejich priority a asociativity. Funkce jsou ukládány znak po znaku a musí být uzavřeny speciálním znakem (zde $), aby bylo při dalším zpracování jasné, že jsou kompletní.

Tabulka 3.1: Převod výrazu sin(x^2) * cos(x) pomocí Shunting Yard algoritmu.

Token	**Akce**	**Výstup (v RPN)**	**Zásobník operátorů**	**Poznámky**
sin	Uložení tokenu do zásobníku		$ n i s	
(	Uložení tokenu do zásobníku		$ n i s (	
x	Přidání tokenu do výstupu	x	$ n i s (	
^	Uložení tokenu do zásobníku	x	$ n i s (^	
2	Přidání tokenu do výstupu	x 2	$ n i s (^	
)	Přesun tokenů ze zásobníku do výstupu	x 2 ^	$ n i s	
	Přesun tokenu ze zásobníku do výstupu	x 2 ^ s i n $		Funkce sin do výstupu.
*	Uložení tokenu do zásobníku	x 2 ^ s i n $	*	
cos	Uložení tokenu do zásobníku	x 2 ^ s i n $	* $ s o c	
(	Uložení tokenu do zásobníku	x 2 ^ s i n $	* $ s o c (	
x	Přidání tokenu do výstupu	x 2 ^ s i n $ x	* $ s o c (	
)	Přesun tokenů ze zásobníku do výstupu	x 2 ^ s i n $ x	* $ s o c	
	Přesun tokenu ze zásobníku do výstupu	x 2 ^ s i n $ x c o s $ *		Funkce cos do výstupu.
konec	Přesun všech tokenů ze zásobníku do výstupu	x 2 ^ s i n $ x c o s $ *		Zpracování zbývajících operátorů.

3.7 Kontrola vytvoření souboru

Je nutné ověřit, zda byla PostScript struktura úspěšně vytvořena, aby bylo zajištěno správné generování výstupu. V případě selhání je proces přerušen a uživatel je informován o chybě.

3.8 Vykresleni grafu

Dále následuje vykreslení grafu, které zahrnuje několik klíčových kroků. Nejprve je vykreslen čtvercový rámec spolu s osami X a Y, poté je přidána mřížka a umístěny popisky s číselnými hodnotami v pravidelných odstupech. Nakonec je vykreslen samotný graf matematické funkce, přičemž hodnoty y jsou postupně vyhodnocovány pomocí reverzní polské notace (RPN).

3.9 Vyhodnocení výrazu v polské reverzní notaci

Vyhodnocení matematického výrazu v polské reverzní notaci (RPN) je klíčový krok při získávání výsledku pro zadanou funkci.

Problémy, které tento krok přináší:

1. **Rozpoznání tokenů**: Program musí správně rozpoznat různé typy tokenů, jako jsou čísla, proměnné, operátory, funkce a speciální znaky.
2. **Zpracování funkcí**: Funkce musí být správně identifikovány, vyhodnoceny a jejich výsledky vloženy zpět do zásobníku.

Řešení:

Rozpoznání tokenů probíhá sekvenčním zpracováním výrazu, kde program analyzuje jednotlivé znaky a na základě jejich vlastností je zařazuje do kategorií, jako jsou čísla, proměnné, operátory nebo speciální znaky. Proces využívá zásobník pro efektivní manipulaci s čísly, operátory a funkcemi. Čísla a proměnné jsou přímo vkládány do zásobníku. Operátory postupně odebírají operandy ze zásobníku, aplikují na ně odpovídající operaci a výsledek ukládají zpět do zásobníku. Na závěr je hodnota na vrcholu zásobníku interpretována jako finální výsledek.

Tabulka 3.2: Zpracování výrazu x, 2, ^, s, i, n, $, x ,c o, s, $, * při $x = 2$.

Token	**Zásobník**	**Poznámky**
x	2	Proměnná x, nahrazena hodnotou 2 a vložena do zásobníku.
2	2, 2	Číslo 2, vloženo přímo do zásobníku.
^	4	Operátor ^ aplikován na 2 a 2. Výsledek $2^2 = 4$ uložen do zásobníku.
s	4	Začátek názvu funkce, znak *s* rozpoznán jako část funkce.
i	4	Pokračování názvu funkce, přidáno k *s*.
n	4	Název funkce dokončen jako sin.
$	-0.7568	Funkce sin aplikována na 4. Výsledek $\sin(4) \approx -0.7568$ uložen do zásobníku.
x	$-0.7568, 2$	Proměnná x, nahrazena hodnotou 2 a vložena do zásobníku.
c	$-0.7568, 2$	Začátek názvu funkce, znak *c* rozpoznán jako část funkce.
o	$-0.7568, 2$	Pokračování názvu funkce, přidáno k *c*.
s	$-0.7568, 2$	Název funkce dokončen jako cos.
$	$-0.7568, -0.4161$	Funkce cos aplikována na 2. Výsledek $\cos(2) \approx -0.4161$ uložen do zásobníku.
$*$	0.3149	Operátor $*$ aplikován na $\sin(4)$ a $\cos(2)$. Výsledek $\sin(4) \cdot \cos(2) \approx 0.3149$ uložen do zásobníku.
Konec	0.3149	Zásobník obsahuje konečný výsledek.

Kapitola 4

Popis implementace

4.1 Hlavní program

Tento program je implementován v souboru `main.c` a zajišťuje zpracování matematických výrazů, jejich validaci a úpravy pro následné vyhodnocení.

Na začátku jsou definovány následující matematické a chybové konstanty:

```
#ifndef M_E
#define M_E 2.71828182845904523536
#endif

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

#define SUCCESS 0
#define ERR_INVALID_ARGUMENTS 1
#define ERR_INVALID_FUNCTION 2
#define ERR_FILE_ERROR 3
#define ERR_INVALID_LIMITS 4
```

Konstanty e a π jsou běžně dostupné přes `math.h` s `#define _USE_MATH_DEFINES`, ale na některých Linuxových systémech to nemusí fungovat. Proto jsou explicitně definovány pro zajištění kompatibility. Chybové kódy slouží k signalizaci různých typů problémů v průběhu zpracování vstupních dat a výpočtů.

4.1.1 Funkce `add_spaces`

Funkce `add_spaces` slouží k předzpracování matematického výrazu, kde upravuje výraz přidáváním mezer, nahrazováním konstant e a π číselnými hodnotami, převodem `exp(x)` na e^x a rozlišováním unárního a binárního operátoru mínus, aby byl snadno analyzovatelný a vhodný pro výpočty.

Při rozpoznání `exp(x)` funkce provede jeho převod na odpovídající zápis s konstantou *e* a exponentem. Proces zahrnuje kontrolu závorek, aby byla správně zpracována i vnořená volání:

```
if (current == 'e' && expression[i + 1] == 'x'
&& expression[i + 2] == 'p') {
    sprintf(e_str, "%.15g", M_E);
    result[j++] = '^';
    while (paren_depth > 0 && ++i < length) {
        result[j++] = expression[i];
    }
}
```

Operátory, jako `+`, `-`, `*`, a `/`, jsou obklopeny mezerami pro snadné rozpoznání jako samostatné tokeny:

```
else if (current == '+' || current == '-'
|| current == '*' || current == '/') {
    result[j++] = current;
}
```

Unární mínus je detekováno a nahrazeno speciálním symbolem $\sim$, který umožňuje jeho jednoznačné odlišení od binárního mínusu:

```
if (current == '-' && (!isdigit(expression[i - 1])
&& expression[i - 1] != ')')) {
    result[j++] = '~';
}
```

Nakonec je výstupní řetězec ukončen nulovým znakem a vrácen jako výsledek.

4.1.2 Funkce `is_valid_function`

Funkce `is_valid_function` ověřuje, zda zadaný matematický výraz obsahuje pouze povolené znaky, operátory a funkce, aby mohl být správně zpracován. Prochází výraz, ignoruje mezery a kontroluje, zda znaky odpovídají povoleným pravidlům. Pokud je nalezen alfanumerický znak, zkontroluje, zda odpovídá povolené funkci ze seznamu:

```
if (isalpha(c)) {
    int valid_function = 0;
    for (int j = 0; allowed_functions[j] != NULL; j++) {
        size_t len = strlen(allowed_functions[j]);
        if (strncmp(&func[i], allowed_functions[j], len) == 0
        && (func[i + len] == '('
        || isspace(func[i + len])
        || func[i + len] == '\0')) {
            valid_function = 1;
            i += len - 1;
            break;
        }
    }
    if (!valid_function) {
        return 0;
    }
}
```

Pokud znak není validní, funkce vrací `0`. Pokud všechny znaky vyhovují, funkce vrací `1`.

4.1.3 Hlavní funkce `main`

Hlavní funkce programu `main` zpracovává vstupní parametry, kontroluje správnost matematického výrazu a jeho omezení, a generuje PostScriptový soubor obsahující graf dané funkce.

Na začátku funkce se kontroluje počet zadaných argumentů. Pokud uživatel nezadá minimálně tři argumenty (funkci, výstupní soubor a volitelně limity grafu), program vypíše chybovou zprávu a skončí s návratovým kódem indikujícím chybu:

```
if(argc < 3) {
    fprintf(stderr, "Error: Missing arguments\n
    Usage: graph.exe <func> <out-file> [<limits>]\n");
    return ERR_INVALID_ARGUMENTS;
}
```

Zadané argumenty jsou následně přiřazeny do proměnných. Funkce je zpracována pomocí `add_spaces`, aby byly přidány mezery kolem operátorů a závorek, a limity grafu jsou volitelné. Dále následuje kontrola, zda funkce obsahuje proměnnou `x`. Pokud ne, program vypíše chybu a skončí:

```
if(strstr(func, "x") == NULL) {
    printf("Error: The function must contain the variable x.\n");
    return ERR_INVALID_FUNCTION;
}
```

Dalším krokem je kontrola, zda funkce obsahuje pouze povolené znaky a definované funkce. Tato kontrola je provedena pomocí `is_valid_function`:

```
if(!is_valid_function(func)) {
    printf("Error: The function contains invalid characters
    or unsupported functions.\n");
    return ERR_INVALID_FUNCTION;
}
```

Pokud byly zadány limity grafu, jsou tyto limity načteny a ověřeny. Limity musí být ve formátu $\langle x_{\texttt{dol}} \rangle : \langle x_{\texttt{hor}} \rangle : \langle y_{\texttt{dol}} \rangle : \langle y_{\texttt{hor}} \rangle$ a $\langle x_{\texttt{dol}} \rangle$ musí být menší než $\langle x_{\texttt{hor}} \rangle$, obdobně $\langle y_{\texttt{dol}} \rangle$ než $\langle y_{\texttt{hor}} \rangle$:

```
if (sscanf(limits, "%lf:%lf:%lf:%lf", &x_min, &x_max,
&y_min, &y_max) != 4) {
    fprintf(stderr, "Error: Invalid format for limits.\n");
    return ERR_INVALID_LIMITS;
}
```

Po validaci vstupů je vytvořen PostScriptový soubor pomocí funkce `create_postscript`. Pokud se soubor nepodaří vytvořit, program skončí s chybou:

```
postscript *ps = create_postscript(outfile, func, x_min, x_max,
y_min, y_max);
if(!ps) {
    fprintf(stderr, "Error: Failed to create PostScript file.
    \n");
    return ERR_FILE_ERROR;
}
```

Funkce následně vykreslí osy, značky, popisky a samotný graf funkce pomocí `draw_square_axis`, `draw_ticks_and_labels` a `draw_graph`. Na konec se PostScriptový soubor uzavře funkcí `close_postscript` a alokovaná paměť pro zpracovanou funkci se uvolní:

```
close_postscript(ps);
free(func);
```

Pokud vše proběhne úspěšně, program vypíše zprávu o úspěšném vytvoření grafu a skončí s návratovým kódem `SUCCESS`.

4.2 PostScript dokument

PostScript je formát pro soubor k zapisování na vykreslování grafiky. Tento modul je implementován v souboru `postscript.c`.

Struktura `postscript` je reprezentována s následujícími vlastnostmi:

- `FILE *file`: Ukazatel na PostScript soubor, do kterého se zapisuje výstup.
- `queue *func`: Fronta obsahující postfixovou reprezentaci matematické funkce.
- `double x_min, x_max, y_min, y_max`: Rozsahy os X a Y.
- `double scale_x, scale_y`: Škálovací faktory pro převod matematických souřadnic na souřadnice obrazovky.

4.2.1 Funkce `create_postscript`

Funkce `create_postscript` vytváří strukturu pro generování PostScript souboru. Nejprve alokuje paměť pro strukturu `postscript` a otevře zadaný soubor k zápisu. Poté inicializuje parametry jako jsou minimální a maximální hodnoty os X a Y, a přepočítá škálovací faktory pro správné vykreslení grafu na plátno.

```
ps->scale_x = POST_SCRIPT_WIDTH / (x_max - x_min);
ps->scale_y = POST_SCRIPT_HEIGHT / (y_max - y_min);
```

Následně nastaví posunutí středu grafu na obrazovce a zapíše hlavičku PostScript souboru:

```
double x_center_offset = (x_min + x_max) / 2.0 * ps->scale_x;
double y_center_offset = (y_min + y_max) / 2.0 * ps->scale_y;

fprintf(ps->file, "%%!PS-Adobe-3.0\n");
fprintf(ps->file, "300 400 translate\n");
fprintf(ps->file, "%.2f %.2f translate\n", -x_center_offset,
-y_center_offset);
```

4.2.2 Funkce `draw_square_axis`

Funkce `draw_square_axis` kreslí hranice a osy grafu. Každá strana čtvercového ohraničení je vykreslena jako samostatná čára:

```
fprintf(ps->file, "newpath\n");
fprintf(ps->file, "%.2f %.2f moveto\n", ps->x_min * ps->scale_x,
ps->y_min * ps->scale_y);
fprintf(ps->file, "%.2f %.2f lineto\n", ps->x_max * ps->scale_x,
ps->y_min * ps->scale_y);
fprintf(ps->file, "stroke\n");
```

4.2.3 Funkce `draw_ticks_and_labels`

Tato funkce přidává značky a popisky na osy X a Y. Intervaly na osách jsou vypočítány následovně:

```
double y_grid_size = (ps->y_max - ps->y_min) / 8;
double x_grid_size = (ps->x_max - ps->x_min) / 8;
```

Každá značka a popisek je vykreslen příslušným PostScript příkazem:

```
fprintf(ps->file, "newpath\n");
fprintf(ps->file, "%.2f %.2f moveto\n", ps->x_min * ps->scale_x,
y_pos);
fprintf(ps->file, "%.2f %.2f lineto\n", ps->x_max * ps->scale_x,
y_pos);
fprintf(ps->file, "stroke\n");

fprintf(ps->file, "%.2f %.2f moveto\n", ps->x_min * ps->scale_x
- 20, y_pos - 3);
fprintf(ps->file, "(%.1f) show\n", y);
```

4.2.4 Funkce `draw_graph`

Funkce `draw_graph` vykresluje matematickou funkci v PostScript souboru. Iteruje přes hodnoty na ose X a počítá odpovídající hodnoty na ose Y. Pokud je hodnota mimo rozsah grafu, přesune pero na novou pozici:

```
for(double x = ps->x_min; x <= ps->x_max; x += 0.001) {
    double y = evaluate_postfix_expression(ps->func, x);
    printf("%.2f %.2f lineto\n", x, y);
    if(y < ps->y_min || y > ps->y_max || isnan(y)) {
        pen_down = 0;
        continue;
    }
```

Jinak pokračuje ve vykreslování linie:

```
if (!pen_down) {
    fprintf(ps->file, "%.2lf %.2lf moveto\n", x_screen, y_screen);
    pen_down = 1;
} else {
    fprintf(ps->file, "%.2lf %.2lf lineto\n", x_screen, y_screen);
}
```

4.2.5 Funkce `close_postscript`

Funkce `close_postscript` uzavírá PostScript soubor a uvolňuje paměť. Na závěr přidá příkaz `showpage` pro dokončení kreslení grafu:

```
fprintf(ps->file, "showpage\n");
fclose(ps->file);
```

4.3 Vyhodnocení matematických výrazů

Vyhodnocení matematických výrazů je implementována v souboru `postfixmath.c`. Tento modul zajišťuje vyhodnocení matematických funkcí a výrazů v postfixové notaci (RPN - Reverse Polish Notation).

4.3.1 Funkce `evaluate_function`

Funkce `evaluate_function` slouží k vyhodnocení jedné matematické funkce pro konkrétní hodnotu. Rozpoznává standardní matematické operace, jako je sinus, kosinus, logaritmus nebo odmocnina, a aplikuje je na hodnotu `x`.

4.3.2 Funkce `evaluate_postfix_expression`

Funkce `evaluate_postfix_expression` slouží k vyhodnocení matematického výrazu zadaného v postfixové notaci (Reverse Polish Notation - RPN). Funkce využívá zásobník pro správu mezivýsledků a umožňuje vyhodnocovat operátory, čísla, funkce a proměnné.

Na začátku funkce je vytvořen zásobník pomocí `stack_create`, který je použit k ukládání čísel a mezivýsledků během vyhodnocování. Pro bezpečné zpracování je následně vytvořena kopie fronty postfixového výrazu, a jelikož se nám procházením fronty bude vymazávat, je následně použita tato kopie.

Dále funkce definuje pomocné proměnné pro čtení čísel a názvů funkcí:

```
char number_buffer[32] = {0};
int number_index = 0;
char function_name[16] = {0};
int function_index = 0;
```

Proměnné `number_buffer` a `function_name` slouží k uchovávání víceznakových hodnot během čtení tokenů. `number_buffer` ukládá jednotlivé znaky čísel pro převod na hodnotu typu `double`, zatímco `function_name` uchovává znaky názvů funkcí, které se po dokončení předávají k vyhodnocení. Indexy `number_index` a `function_index` ukazují pozici pro zápis dalšího znaku.

Hlavní část funkce iteruje přes jednotlivé tokeny ve frontě. Každý token je zpracován podle svého typu. Pokud je token číslice nebo desetinná tečka, je uložen do bufferu `number_buffer`. Po dokončení čtení je číslo převedeno na typ `double` a vloženo do zásobníku:

```
if (isdigit(token) || token == '.') {
    number_buffer[number_index++] = token;
}
else if (number_index > 0 && (strchr("+-*/^~$", token) ||
isalpha(token) || token == 'x')) {
    number_buffer[number_index] = '\0';
    double num = atof(number_buffer);
    if (is_unary) {
        num = -num;
        is_unary = 0;
    }
    stack_push(s, &num);
    memset(number_buffer, 0, sizeof(number_buffer));
    number_index = 0;
}
```

Pokud je token `x`, funkce vloží hodnotu `x_value` do zásobníku. Pokud je aktivní unární

mínus, je hodnota negována:

```
if (token == 'x') {
    if (is_unary) {
        x_value = -x_value;
        is_unary = 0;
    }
    stack_push(s, &x_value);
}
```

Tokeny odpovídající funkcím jsou zpracovány pomocí bufferu `function_name`. Jakmile je funkce ukončena znakem `$`, její název je předán funkci `evaluate_function`, která ji aplikuje na hodnotu ze zásobníku:

```
else if (token == '$') {
    function_name[function_index] = '\0';
    if (!stack_pop(s, &a)) {
        queue_free(&expression_copy);
        stack_free(&s);
        return NAN;
    }
    if (is_unary) {
        a = -a;
        is_unary = 0;
    }
    result = evaluate_function(function_name, a);
    stack_push(s, &result);
    memset(function_name, 0, sizeof(function_name));
    function_index = 0;
}
```

Operátory, jako jsou `+`, `-`, `*`, `/` a `^`, vyžadují dvě hodnoty ze zásobníku. Operátor aplikuje odpovídající operaci a výsledek uloží zpět do zásobníku:

```
else if (strchr("+-*/^", token)) {
    if (!stack_pop(s, &b) || !stack_pop(s, &a)) {
        queue_free(&expression_copy);
        stack_free(&s);
        return NAN;
    }
    switch (token) {
        case '+': result = a + b; break;
        case '-': result = a - b; break;
        case '*': result = a * b; break;
        case '/': result = a / b; break;
        case '^': result = pow(a, b); break;
    }
    stack_push(s, &result);
}
```

Po zpracování všech tokenů zůstane v zásobníku jediná hodnota, která je výsledkem výrazu. Na konci funkce jsou uvolněny všechny dočasné struktury, aby se předešlo únikům paměti.

4.4 Shunting yard algoritmus

Převod výrazu do polské reverzní notace je prováděn pomocí Shunting Yard algoritmu. Shunting yard algoritmus je **implementována v souboru shuntingyard.c** Tento algoritmus postupně zpracovává jednotlivé tokeny výrazu a podle jejich povahy rozhoduje, zda je zařadí do výstupní fronty, nebo je uloží do zásobníku. Operátory jsou v zásobníku řazeny podle jejich priority a asociativity, což zajišťuje správné pořadí ve výstupní RPN. Funkce jsou rozpoznávány a dočasně ukládány do zásobníku, do- kud nejsou jejich argumenty správně zpracovány. Pro rozeznaní jsou proto ukončeny znakem $. Závorky jsou správně párovány a odstraněny v okamžiku, kdy je jejich obsah kompletně zpracován. Unární operátory, jako je mínus, jsou označeny speciálním symbolem, aby byly jednoznačně odlišeny od binárních operátorů.

4.4.1 Funkce `strdup`

Funkce `strdup` slouží k vytvoření duplikátu zadaného řetězce. Nejprve určí délku vstupního řetězce a následně alokuje dostatek paměti pro jeho kopii. Po alokaci funkce zkopíruje obsah původního řetězce do nově alokované paměti:

```
size_t len = strlen(s) + 1;
char *dup = malloc(len);
if (!dup) return NULL;
memcpy(dup, s, len);
```

Tato implementace je součástí projektu z důvodu, že některé distribuce Linuxu nemusí standardně poskytovat funkci `strdup`.

4.4.2 Funkce `precedence`

Funkce `precedence` určuje prioritu matematického operátoru. Vrací hodnotu odrážející úroveň priority, kde vyšší číslo znamená vyšší prioritu (viz tabulka).

Tabulka 4.1: Priorita operátorů

Operátor	Priorita
+ nebo -	1
* nebo /	2
^	3

4.4.3 Funkce `is_left_associative`

Funkce `is_left_associative` kontroluje, zda je matematický operátor levostranně asociativní. Operátor `^` (mocnina) není levostranně asociativní, protože jeho pravostranná asociativita umožňuje správné vyhodnocení výrazů, jako je `2 ^ 3 ^ 2`. Pokud by `^` byl levostranně asociativní, výraz by byl interpretován jako `(2 ^ 3) ^ 2`, což není matematicky správné. Pravostranná asociativita zajistí, že mocniny se počítají zprava doleva, tedy `2 ^ (3 ^ 2)`.

4.4.4 Funkce `is_operator`

Funkce `is_operator` kontroluje, zda zadaný znak představuje matematický operátor. Rozpozná následující operátory: `+`, `-`, `*`, `/`, `^`.

4.4.5 Funkce `shunting_yard`

Funkce `shunting_yard` slouží k převodu matematického výrazu z infixové notace do postfixové notace (RPN - Reverse Polish Notation). Tento algoritmus využívá zásobník pro správu operátorů a funkcí a výstupní frontu pro ukládání výsledných tokenů.

Na začátku funkce je vytvořen zásobník pomocí `stack_create`, který bude použit k ukládání operátorů a funkcí během zpracování výrazu. Poté je vstupní řetězec zkopírován pomocí `strdup`, aby bylo možné bezpečně zpracovávat jeho obsah bez rizika změny originálního výrazu.

Následně je vstupní řetězec tokenizován, tedy rozdělen na části podle mezer, a každý token je postupně analyzován. Pokud je token číslo, je jeho obsah přímo přidán do výstupní fronty pomocí `queue_enqueue`:

```
if (isdigit(token[0]) || (token[0] == '-'
&& isdigit(token[1]))) {
    for (size_t i = 0; i < strlen(token); i++) {
        char num_char = token[i];
        if (!queue_enqueue(output, &num_char)) {
            free(expr_copy);
            stack_free(&holding_stack);
            return;
        }
    }
}
```

Pokud je token operátor, algoritmus porovnává jeho prioritu s operátory, které již jsou v zásobníku. Pokud má operátor na vrcholu zásobníku vyšší nebo stejnou prioritu, je přenesen do výstupní fronty:

```
else if (is_operator(token[0])) {
    char op1 = token[0];
    while (holding_stack->sp >= 0) {
        char top_op;
        if (!stack_peek(holding_stack, &top_op)) break;
        if (is_operator(top_op) &&
        ((precedence(top_op) > precedence(op1)) ||
        (precedence(top_op) == precedence(op1)
        && is_left_associative(op1)))) {
            stack_pop(holding_stack, &top_op);
            if (!queue_enqueue(output, &top_op)) {
                free(expr_copy);
                stack_free(&holding_stack);
                return;
            }
        } else {
            break;
        }
    }
    stack_push(holding_stack, &op1);
}
```

Závorky jsou zpracovány odlišně. Otevírací závorky jsou jednoduše uloženy do zásobníku. Uzavírací závorky způsobí přesun všech operátorů z vrcholu zásobníku do výstupní fronty, dokud není nalezena odpovídající otevírací závorka:

```
else if (strcmp(token, "(") == 0) {
    char paren = '(';
    stack_push(holding_stack, &paren);
} else if (strcmp(token, ")") == 0) {
    char top_op;
    while (stack_pop(holding_stack, &top_op) && top_op != '(') {
        if (!queue_enqueue(output, &top_op)) {
            free(expr_copy);
            stack_free(&holding_stack);
            return;
        }
    }
}
```

Po zpracování všech tokenů jsou zbývající operátory v zásobníku přesunuty do výstupní fronty:

```
char top_op;
while (stack_pop(holding_stack, &top_op)) {
    if (top_op == '(' || top_op == ')') {
        break;
    }
    if (!queue_enqueue(output, &top_op)) {
        free(expr_copy);
        stack_free(&holding_stack);
        return;
    }
}
```

Na závěr funkce uvolní veškerou alokovanou paměť, včetně kopie výrazu a zásobníku, aby nedocházelo k únikům paměti. Pro zvýšení bezpečnosti jsou ukazatele nastaveny na `NULL`.

4.4.6 Funkce `is_function`

Funkce `is_function` kontroluje, zda zadaný řetězec odpovídá platné matematické funkci.

4.5 Fronta

Fronta je datová struktura, která pracuje na principu ”First In, First Out”. To znamená, že první přidaný prvek je také první, který se odebere.
Fronta je implementována v souboru **queue.c** a je **reprezentována** strukturou `queue`, která má následující vlastnosti:

- `uint size`: Maximální počet položek, které může fronta uchovávat.
- `uint item_size`: Velikost každé položky v bytech.
- `int first`: Index první položky ve frontě.
- `int last`: Index poslední položky ve frontě.
- `int count`: Počet aktuálních položek ve frontě.
- `void *items`: Ukazatel na paměť pro uložení položek fronty.

Fronta podporuje následující operace:

- `queue_create`: Vytváří novou frontu, alokuje paměť a inicializuje její parametry.
- `queue_enqueue`: Přidává novou položku na konec fronty.
- `queue_dequeue`: Odstraňuje položku ze začátku fronty.
- `queue_free`: Uvolňuje veškerou paměť alokovanou pro frontu.
- `queue_copy`: Vytváří kopii existující fronty.

4.6 Zásobník

Zásobník je datová struktura, která pracuje na principu "Last In, First Out". To znamená, že poslední přidaný prvek je první, který se odebere.

Zásobník je implementován v souboru **stack.c** a je **reprezentován** strukturou `stack`, která má následující vlastnosti:

- `int sp`: Ukazatel vrcholu zásobníku, inicializovaný na -1.
- `uint size`: Maximální počet položek zásobníku.
- `uint item_size`: Velikost každé položky v bytech.
- `void *items`: Ukazatel na paměť pro uložení položek.

Zásobník podporuje následující operace:

- `stack_create`: Vytváří nový zásobník, alokuje paměť a inicializuje jeho parametry.
- `stack_push`: Přidává novou položku na vrchol zásobníku.
- `stack_pop`: Odstraňuje položku z vrcholu zásobníku.
- `stack_peek`: Čte položku z vrcholu zásobníku, aniž by ji odstranila.
- `stack_free`: Uvolňuje veškerou paměť alokovanou pro zásobník.

Kapitola 5

Uživatelská příručka

5.1 Postup přeložení programu

5.1.1 Linux

Na platformě Linux **je potřeba mít nainstalovaný kompilátor gcc a nástroj make**. Přejděte do adresáře, kde máte uložený zdrojový kód programu, a spusťte příkaz:

```
X:\>make
```

5.1.2 Windows

Pro Windows použijte soubor makefile.win, který je součástí zdrojového kódu. Pro přeložení **musí být nainstalovaný gcc a MinGW**. Otevřete příkazový řádek a přejděte do adresáře, kde máte uložený zdrojový kód. Poté spusťte:

```
X:\>mingw32-make -f makefile.win
```

5.2 Spuštění programu

Program **je ovládán prostřednictvím příkazové řádky**. Uživatel zadává matematickou funkci, kterou chce analyzovat a vykreslit, název výstupního souboru (ve formátu PostScript), a volitelný parametr pro určení rozsahu, ve kterém má být funkce vykreslena. Příklad spuštění programu je následující:

```
X:\>graph.exe sin(x^2)*cos(x) postscript.ps
```

Funkce může být zadána buď bez uvozovek, nebo s nimi, a to jak s mezerami, tak bez nich.

Například:

```
X:\>graph.exe "sin(x ^ 2 ) * cos(x )" postscript.ps
```

Pokud chcete omezit rozsah hodnot pro definiční obor a obor hodnot, můžete přidat volitelný parametr, který je ve formátu $\langle x_{\text{dol}} \rangle : \langle x_{\text{hor}} \rangle : \langle y_{\text{dol}} \rangle : \langle y_{\text{hor}} \rangle$:

```
X:\>graph.exe "sin(x ^ 2 ) * cos(x )" postscript.ps -10:10:-5:5
```

Pokud nejsou rozsahy zadané, program použije implicitní hodnoty pro $x \in \langle -10; 10 \rangle$ a $y \in \langle -10; 10 \rangle$.

Funkce, znaky a konstanty v uživatelském vstupu

Vstupní řetězec, který uživatel zadává, může obsahovat různé matematické funkce, operátory a konstanty, které jsou podporovány programem. Následující seznam uvádí povolené funkce, operátory a konstanty, které je možné použít:

Matematické funkce:

Tabulka 5.1: Seznam matematických funkcí a jejich význam.

Funkce	Význam
`sin`	Sinus
`cos`	Kosinus
`tan`	Tangens
`asin`	Inverzní sinus
`acos`	Inverzní kosinus
`atan`	Inverzní tangens
`sinh`	Hyperbolický sinus
`cosh`	Hyperbolický kosinus
`tanh`	Hyperbolický tangens
`ln`	Přirozený logaritmus
`log`	Dekadický logaritmus
`sqrt`	Odmocnina
`abs`	Absolutní hodnota
`exp`	Výpočet *e*^zadaná hodnota

Operátory:

Tabulka 5.2: Seznam operátorů a jejich význam.

Operátor	Význam
`-`	Unární mínus
`+`	Sčítání
`-`	Odčítání
`*`	Násobení
`/`	Dělení
`^`	Umocnění

Konstanty a proměnné:

Tabulka 5.3: Seznam konstant a proměnných.

Konstanty a proměnné	Význam
`pi`	Číslo Pí
`e`	Eulerovo číslo
`1E0`	Reálné číslo ve formátu 1E0 (vědecká notace)
`x`	Nezávislá proměnná

5.3 Výstupy programu

Výstupy programu mohou být v různých scénářích závislé na správnosti zadaných argumentů, funkcí a formátu vstupů. Níže jsou uvádeny výstupy pro různé situace:

Tabulka 5.4: Výstupy programu pro různé scénáře a jejich návratové hodnoty.

Scénář	**Vstupní příkaz**	**Výstup programu**	**Návratová hodnota**
Úspěšné vytvoření grafu	`graph.exe "sin(x)" output.ps -10:10:-1:1`	`Graph successfully generated in file: output.ps`	`0`
Chybějící argumenty	`graph.exe "sin(x)"`	`Error: Missing arguments Usage: graph.exe <func> <out-file> [<limits>]`	`1`
Funkce neobsahuje proměnnou x	`graph.exe "sin()" output.ps -10:10:-1:1`	`Error: The function must contain the variable x.`	`2`
Nepovolené znaky ve funkci	`graph.exe "invalid_function(x)" output.ps -10:10:-1:1`	`Error: The function contains invalid characters or unsupported functions.`	`2`
Nesprávný formát limitů	`graph.exe "sin(x)" output.ps 10;-10;-1;1`	`Error: Invalid format for limits.`	`4`
Špatné pořadí limitů	`graph.exe "sin(x)" output.ps 10:5:-1:1`	`Error: x_min must be less than x_max.`	`4`
Selhání při vytváření souboru	`graph.exe "sin(x)" restricted/ output.ps -10:10:-1:1`	`Error: Failed to create PostScript file.`	`3`

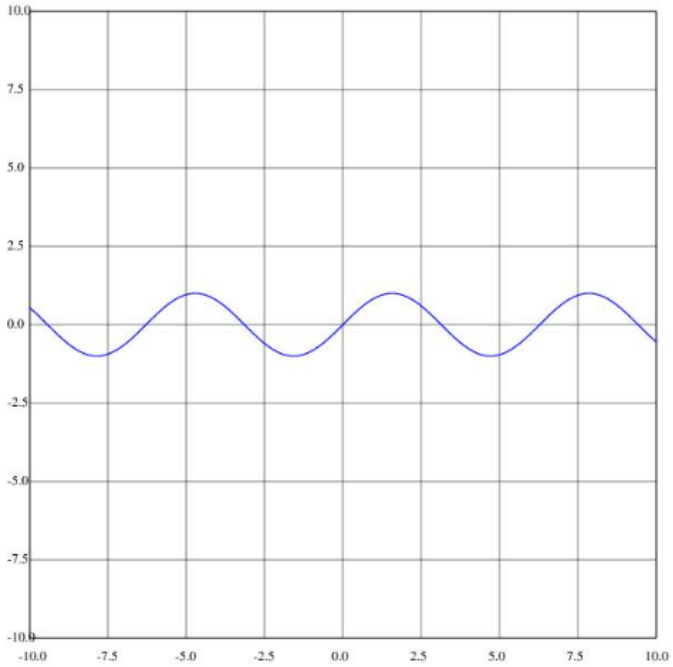
5.4 Ukázky běhu programu pro různá vstupní data

Výsledné `.ps` soubory byly zobrazeny pomocí online nástroje na této stránce: https://psviewer.org/onlineviewer.aspx.

Příklad 1 – Jednoduchý matematický výraz:

```
X:\>graph.exe "sin ( x )" postscript.ps
```

Tento příkaz vygeneruje graf funkce $y = \sin(x)$ (viz. obrázek 5.1) na intervalu $x \in \langle -10; 10 \rangle$ a $y \in \langle -10; 10 \rangle$.



Obrázek 5.1: Graf funkce $y = sin(x)$.

Příklad 2 – Komplexní matematický výraz s limity:

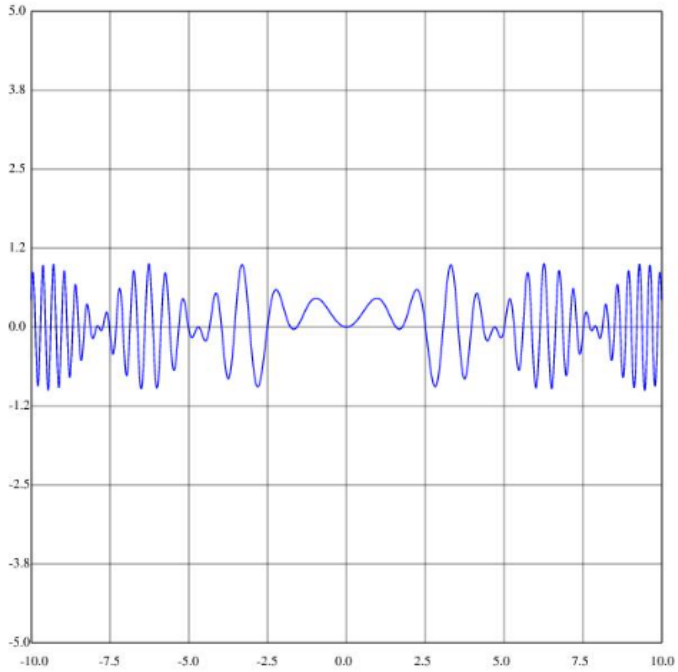
```
X:\>graph.exe "sin(x ^ 2 ) * cos(x )" postscript.ps -10:10:-5:5
```

Tento příkaz vygeneruje graf funkce $y = \sin(x^2) \cdot \cos(x)$ (viz. obrázek 5.2) pro hodnoty $x \in \langle -10; 10 \rangle$ a $y \in \langle -5; 5 \rangle$.

Příklad 3 – Funkce bez uvozovek:

```
X:\>graph.exe sin(x^2)*cos(x) postscript.ps -10:10:-5:5
```

Tento příkaz vykreslí stejný graf (viz. obrázek 5.2) jako předchozí příklad, ale bez použití uvozovek.



Obrázek 5.2: Graf funkce $y = sin(x^2) * cos(x)$ na intervalu $x \in \langle -10; 10 \rangle$ a $y \in \langle -5; 5 \rangle$.

Kapitola 6

Závěr

Tato semestrální práce se zaměřila na implementaci konzolové aplikace, která analyzuje a vizualizuje matematické funkce. Hlavním cílem bylo vytvořit nástroj, jenž dokáže zpracovávat matematické výrazy a následně generovat grafy těchto funkcí ve formátu PostScript.

Všechny požadavky práce byly splněny. Program je schopen načíst matematickou funkci, provést její analýzu, přidat mezery mezi operátory, funkcemi a závorkami a vygenerovat PostScript soubor pro její zobrazení. Byla implementována validace zápisu, která zajišťuje správné zpracování výrazů a generování chybových hlášek v případě neplatného zápisu, což je klíčové pro zajištění správného fungování programu.

V budoucnu by bylo možné práci vylepšit přidáním širšího spektra matematických funkcí, než jaké jsou aktuálně implementovány. Dalším možným vylepšením je optimalizace kódu, protože některé kontroly chyb mohou být prováděny vícekrát, což by mohlo vést k zbytečným výpočtům. Dále by bylo žádoucí přidat funkci pro zobrazení neexistujících bodů nebo neexistujících funkcí, což by posunulo aplikaci o úroveň výš. Tento přístup by také umožnil rozšířit aplikaci na pokročilejší výpočty a grafické zobrazení.

Během práce na této semestrální práci jsem měl možnost prozkoumat širokou problematiku analýzy matematických funkcí a grafického zobrazení. Práce mě velmi bavila a byla pro mě zajímavým způsobem, jak si prohloubit znalosti v oblasti programování, nejen v programovacím jazyce C, ale i v psaní kvalitní dokumentace. V průběhu semestru jsem implementovaný nástroj využíval i při řešení dalších úloh, což potvrdilo jeho praktickou použitelnost a funkčnost. Jsem rád, že se mi podařilo realizovat všechny požadavky, a cítím, že tato práce byla úspěšně dokončena.