TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta strojní

Katedra Aplikované Kybernetiky

BAKALÁŘSKÁ PRÁCE

2007

Petr Novák

TECHNICKÁ UNIVERZITA V LIBERCI

FAKULTA STROJNÍ

Studijní obor 2301R030

VÝROBNÍ SYSTÉMY

se zaměřením na:

INŽENÝRSKOU INFORMATIKU

REXX KNIHOVNA PRO VIZUALIZACI MĚŘENÝCH DAT

REXX LIBRARY FOR VISUALIZATION OF MEASURED DATA

Petr Novák

Vedoucí bakalářské práce: Ing. Michal Moučka, Ph.D.

Rozsah stran: 33
Počet obrázků: 4
Počet fragmentů: 5

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta strojní

Katedra aplikované kybernetiky Studijní rok: 2006/07

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jméno a příjmení: **Petr NOVÁK**

Studijní program: **B2341 Strojírenství**

Obor: **2301R030 Výrobní systémy**

Zaměření: **Inženýrská informatika**

Ve smyslu zákona č. 111/1998 Sb. o vysokých školách se Vám určuje bakalářská práce na téma:

REXX knihovna pro vizualizaci měřených dat

Zásady pro vypracování:
(uveďte hlavní cíle bakalářské práce doporučené metody pro vypracování)

1. Seznamte se s programovacím jazykem REXX.
2. Navrhněte dynamickou knihovnu (Dynamic-link library, DLL) pro grafickou vizualizaci měřených dat tak, aby její funkce byly volatelné z jazyka REXX. Knihovna musí obsahovat funkce pro zobrazení grafů, nastavení měřítek a popisů datových os.
3. Realizujte knihovnu. Knihovnu implementujte v jazyku C s využitím programového rozhraní WIN32.
4. Funkčnost a správnost navrženého řešení ověřte na jednoduché vzorové REXX aplikaci.

ANOTACE

Bakalářská práce se zabývá popisem vytvoření aplikace pro vykreslování grafů a následné použití pro vizualizaci naměřených hodnot. Dále pak popisem použitých programovacích jazyků a sice jazyka C s využitím rozhraní WIN32 API a Objektově orientovaného jazyka REXX.

ANNOTATION

Bachelor work deals with description of creating application for graph's drawing and subsequent using for visualization of measured values. The other way is about description used programmer's languages, there is mentioned C language with WIN32 API interface and Object oriented REXX language.

Obsah

1 Úvod

Hlavní náplní bakalářské práce je vytvořit funkční aplikaci pro vykreslování grafů ze souboru, včetně možnosti volitelného měřítka, popisků os, popisku hlavního okna, přidávání grafů. Aplikace má být vytvořena v prostředí programovacího jazyku C. Pro ovládání aplikace bude vytvořena dynamická knihovna funkcí pro jazyk REXX.

Práce je rozdělena na tři základní části. V první části, s podtitulem Programovací prostředí, se seznámíme s historií jazyků REXX a C. Seznámíme se se základem programování v REXXu a ukážeme si jej v praxi na dvou jednoduchých příkladech. Vysvětlíme si pojem WIN32API včetně jeho hlavních charakteristik a předností, popíšeme si princip vytvoření hlavního okna, volání zpráv a ostatní použité propriety.

Druhá část, nesoucí název Vytváření aplikace v OS Windows XP, je tématicky zaměřena na shrnutí vývoje aplikace včetně popisu většiny jeho funkcí s ukázkou vybraných částí zdrojového kódu a jeho převedení na knihovnu. Následuje použití knihovny jazykem REXX.

Poslední část práce, Závěr, shrnuje dosažené výsledky a hodnotí použité metody.

2 Programovací prostředí

2.1 Historie jazyka C

Vývoj jazyka C započali v roce 1969 Brian W. Kernighan a Dennis M. Ritchie v Bellových laboratořích zapůjčených firmou AT&T a to jako lehce modifikovatelný a účinný programovací jazyk [10]. Název byl zvolen podle staršího jazyka „B", z kterého byla použita většina funkcí a vlastností tohoto jazyka. Původně byl určen především pro operační systém UNIX (operační systém UNIX 7 byl téměř celý napsán v tomto jazyce), ale na počátku osmdesátých let s nástupem platformy IBM PC se začínají objevovat kompilátory i pro další platformy a stává se tak univerzálním jazykem.

V roce 1973 se rozvinul o objektově orientované programování a vznikl jazyk C++. Jazyk C byl definován v knize „The C Programming Language" rovněž od autorů Kernighana a Ritchieho a v roce 1983 American National Standards Institute sestavil komisi X3J11, která měla vytvořit standardní specifikaci tohoto jazyka. Proces byl dokončen o šest let později a vešel v platnost jako ANSI X3.159-1989 „Programming Language C" [9]. Tato verze jazyka je často označována jako ANSI C. Cílem standardizace bylo především zahrnutí vylepšeného preprocesoru a funkčních prototypů vypůjčených z jazyka C++.

K dispozici jsou volně dostupné implementace jazyka. Například na oficiálních stránkách společnosti Microsoft - Visual Studio Express 2005. K mé práci bylo právě použito toto vývojové prostředí, střídavě se starší verzí Visual Studio 6. Dále lze bez problémů použít OpenWatcom, který je velmi nenáročnou a v mnoha ohledech lepší variantou.

2.2 Historie jazyka REXX

Programovací jazyk REXX byl vytvořen v touze po získání lehce ovladatelného, jednoduchého a zároveň účinného programovacího jazyka pro společnost IBM. Vývojářem se stal Mike Cowlishaw a zvolil sympatický název REX [11]. Důvodem toho byla implementace většiny vlastností ze starších jazyků EXEC 2 a PL/I. Důvodem proč byl jazyk přejmenován z původního názvu REX na REXX (REstructured eXtended eXekutor) byla dřívější registrace jiného projektu se shodným názvem.

První specifikace byla zaznamenána 29.3.1973 a již obsahovala tři vzorové programy psané právě v prostředí REXXu. První vlastní implementace byla vytvořena jako volně dostupná přes interní síť společnosti IBM na konci osmdesátých let. Právě s rozšířením této sítě se jazyk stal velmi populárním, protože uživatelé si mohli vzájemně vyměňovat postřehy a zkušenosti a společně navrhovat vylepšení jazyka. Výhodou byla i kooperace společnosti IBM, která v rámci unifikace jazyka přepracovala původní ARG instrukce na novější ARGS, což si vyžádalo aktualizace stovek tisíc programů psaných v REXXu. Výsledkem této přímé odezvy společnosti se jazyk velmi rychle vyvíjel ke spokojenosti uživatelů a vývojářů.

V roce 1984 byla vydána první implementace jazyka společností QUERCUS SOFTWARE pro ostatní operační systémy, tedy i pro DOS. Následoval příchod prvního kompilátoru vyvinutého ve Vídeňských laboratořích společnosti IBM v roce 1989. Následující rok nastartoval sérii vědeckých symposií pro vývojáře a uživatele, organizovanou Stanfordským lineárním centrem v Kalifornii. V roce 1990 se stává pevnou částí operačního systému OS/2 a to včetně komponent pro ovládání multimédií. V současné době existuje množství interpretů pro všechny operační systémy. Standardní REXX byl rozšířen o konstrukce objektově orientovaného programování (OOP) – vznikl jazyk Object REXX.

K dispozici jsou verze komerční stejně jako verze volně dostupné na oficiálních stránkách společnosti IBM. Mezi volně stažitelný software patří například Open ObjectREXX či Regina REXX, které se liší v některých vlastnostech (knihovnách).

2.3 Programování v jazyku REXX

Programy v jazyku REXX jsou snadno přenositelné do prostředí jiných operačních systémů (důkaz najdete na webovské stránce tvůrce jazyka M. F. Cowlishawa) [11]. Výrazovými prostředky jazyka lze zapsat i složitý algoritmus krátkým programem. Krátké programy mohou být užitečné jako prototypy.

Je vhodný pro výuku programování. Pokusem o ucelený důkaz možností jazyka při popisu, definici a kódování algoritmů je Album algoritmů a technik [14]. Je ideálním nástrojem pro vývoj a testování algoritmů. Začínající programátor bude považovat asociativní pole a interpretaci za přirozené prostředky. Rada pro začínající programátory: Program napište nejprve v jednoduchém pseudojazyce a teprve potom jej převeďte do jazyka, který používáte [15]. Pro řadu lidí je rozhodně ideálním pseudojazykem, právě kvůli jeho vlastnostem a možnostem užití. Pro ukázku následují dva velmi triviální příklady.

Příklad č.1:

Tato aplikace vypíše text a zjistí kolikrát je inkrementována proměnná ***S*** až do splnění podmínky, výsledek uloží do proměnné *E* a následně vypíše na obrazovku.

```
/*testik*/ /* způsob zápisu poznámek */
say "ZDAR" /* vypíše na obrazovku obsah ohraničený uvozovkami */
S=15 /* přiřazení hodnot proměnným S a E */
E=0
do while S <=20 /* cyklus dělej dokud proměnná S nebude menší nebo rovna
hodnotě 20 */
        S=S+1 /* inkrementace proměnných v těle cyklu */
        E=E+1
end /* ukončení cyklu */
say "POCET JE " E
```

Příklad č.2:

Tento program vypíše pozdrav na obrazovku a do proměnné ***n*** načte počet čísel. Uživatel postupně zadává operaci součet nebo rozdíl a požadované hodnoty. Výsledkem je výpis počtu zadaných čísel a výsledná hodnota.

```
/*soucet n cisel*/
say "ZDAREK"
S=0
E=0
i=1
say "zadej pocet cisel"
pull n /* funkce pro načtení znaku z klávesnice, ekvivalent fce. scanf jazyka C */
SAY "zadej "i". cislo"
        pull D
        do while i<n
        SAY "vyber proces"
        pull P
        if p= '+' then
                S=S+D
        if p= '-' then
                S=S-D
        SAY "zadej "i+1". cislo"
        pull D
        i=i+1
end
SAY "POCET zadanych cisel " n
SAY "vysledek " S
```

Z uvedených příkladů je zřejmá jednoduchost syntaxe jazyka REXX. Lze si všimnout, že funkčnost aplikací neovlivní ani fakt, zda příkazy píšeme písmem velkým či malým. Velmi zajímavé je, že zdrojový kód lze psát v kterémkoliv textovém editoru a aplikaci spouštět přes příkazovou řádku, kde můžeme dokonce zjistit, který řádek obsahuje chybu.

2.4 Popis programovacího rozhranní WIN32API

Pod zkratkou API se ukrývá Application Programming Interface což v překladu znamená aplikační programovací rozhraní [6]. WIN32 API bylo vyvinuto společností Microsoft pro vytváření aplikací pod operační systém MS Windows. Jedná se o soubor všech přístupných funkcí, se kterými může námi vytvářená aplikace pracovat. Dále obsahuje definice datových struktur a typů. Na rozdíl od operačního systému MS-DOS, ve Windows API neexistují žádné vstupní body a čísla služeb, ale každá služba se volá jménem jako funkce, a proto se často služby OS Windows nazývají jednoduše funkce.

Pod pojmem WIN32API si tedy můžeme představit programovací rozhraní, které využíváme ke komunikaci s operačním systémem Windows při tvorbě aplikací založených na grafickém rozhraní. Pracuje se systémem 32-bitových čísel/parametrů ale z hlediska zpětné kompatibility obsahuje i několik 16-bitových funkcí. V dnešní době obsahuje nepřebernou škálu funkcí pracujících například s grafikou (GDI), se zvukem (MCI), ovládacím zařízením a sítí. Používá se velmi často v kombinaci s DirectX či OpenGL [7].

Výhodou Windows API je podobnost základních prvků pro jednotlivé verze Windows. Můžeme tedy říci, že většina programů vytvořených pro Windows 3.11 bude pracovat správně i pod novějšími operačními systémy ze stáje Microsoft, například pod Windows XP. S každým nově nastupujícím operačním systémem narůstá množství nově definovaných funkcí a datových typů, některé nepoužívané funkce zanikají nebo jsou pouze předělány na požadované parametry. Z tohoto důvodu je zbytečné si vše pamatovat nazpaměť, dá se říci že je to v podstatě nemožné vzhledem k množství funkcí. Proto existují jak základní příručky pro práci s tímto rozhraním, tak na oficiálních stránkách Microsoft můžeme nalézt tzv. MSDN knihovnu která zahrnuje definice všech funkcí, prototypů, zpráv a ostatních náležitostí právě k tomuto rozhraní. Další variantou jsou implementované nápovědy (help), sloužící pro snadnější a rychlejší porozumění systému které jsou součástí každého solidního kompilátoru.

2.4.1 Hlavní vstupní bod programu

Každý standardní program ve Windows musí mít jednu tzv. hlavní funkci, pro Windows nese označení **WinMain.** Funkce má tuto definici:

WinMain(HINSTANCE ***hInst***, HINSTANCE ***hPrevInst***, LPSTR ***szCmdLine***, int ***nCmdShow***)

> ***hInst*** – handle instance, které nám "přidělí" operační systém při spuštění programu. Jde o 32-ti bitové číslo jednoznačně identifikující náš běžící program mezi ostatními programy a vlákny v systému
>
> ***hPrevInst*** – tento parametr je zde pouze pro zpětnou kompatibilitu zdrojového kódu s Win16 programy. Pokud bude aplikace spuštěna vícekrát, bude obsahovat parametr právě handle předchozí spuštěné instance
>
> ***szCmdLine*** – ukazatel na řetězec obsahující parametry příkazové řádky (command line)
>
> ***nCmdShow*** – hodnota určující styl zobrazení okna, např. zda se má okno spustit minimalizované nebo v normálním stavu

2.4.2 Vytvoření hlavního okna

Definice okna: Okno je obdélníková oblast na obrazovce, která přebírá vstup od uživatele a zobrazuje výstup v podobě textu a grafiky. Tato aplikace používá jak okno hlavní (mateřské), tak i okna vedlejší (dětská) pro vykreslování grafů. V této kapitole si popíšeme základní pojmy okna a jeho vytvoření. Při vytváření aplikace pro OS Windows musíme vždy přiřadit hlavni knihovnu: **#include <windows.h>**

Hlavní části kódu okna:

1) vstupní bod (WinMain)
2) třída okna, registrace třídy okna
3) vytvoření okna
4) smyčka zpráv

Třída okna je procedura, která provádí zpracování zpráv daného okna a pomocí níž stanovíme vlastnosti okna (menu, ikona, kurzor, barva pozadí, atd.). Po nastavení všech potřebných parametrů třídy okna musíme tuto třídu zaregistrovat. Tato akce se provede voláním funkce RegisterClass (*&wc*), kde dojde k přiřazení ukazatele (pointer) na strukturu WNDCLASSEX do proměnné *wc*.

K vytvoření okna dochází až na základě volání funkce pro vytvoření okna – CreateWindow. Funkce vrací handle (manipulátor) okna a ukládá jej do hWnd typu HWND. Pomocí tohoto manipulátoru se lze na okno odkazovat (vykreslování grafů v okně). Tato funkce obsahuje určující parametry okna, jako například jeho velikost a pozici na monitoru. Důvodem je snadná změna parametrů, například u MDI oken, kdy na každé z nich jdou kladeny rozdílné požadavky, není třeba pro každé z nich provádět novou registraci třídy.

Smyčka zpráv je procedura, která se stará o třídění zpráv a jejich přidání do fronty. Pracuje do té doby, než dojde ke zpracování zprávy WM_DESTROY, ukončující aplikaci. Je umístěna na konec kódu programu, funkce a zprávy poté píšeme právě mezi tuto smyčku a proceduru hlavního okna. Následuje fragment kódu programu pro vytvoření okna:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// add 1) Hlavní funkce
int WINAPI WinMain( HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR szCmdLine, int nCmdShow ) {
//deklarace proměnných
  MSG       msg;
  WNDCLASS  wc;
  BOOL      rc;
  int       x, y, cx, cy;
```

```
//add 2) Nastavení parametrů třídy okna
  wc.style =          CS_HREDRAW | CS_VREDRAW  //styl vykreslování okna
  wc.lpfnWndProc =    (WNDPROC)MainWndProc; // procedura okna
  wc.cbClsExtra =     0; // rezervovaný prostor třídy
  wc.cbWndExtra =     0; // rezervovaný prostor třídy
  wc.hInstance =      hInst; // manipulátor(handle) instance
  wc.hIcon =          LoadIcon( hInst,MAKEINTRESOURCE(IDI_ICON1) ); // volba ikony
  wc.hCursor =        (HCURSOR)LoadCursor(NULL, IDC_ARROW); //volba kurzoru
  wc.hbrBackground = (HBRUSH)(COLOR_APPWORKSPACE + 1); // barva pozadí
  wc.lpszMenuName = NULL; // název menu okna
  wc.lpszClassName = "UC_GRAPH"; // název třídy okna
//add 2) Registrace třídy okna
  rc = RegisterClass( &wc ); // registrace třídy mateřského okna
       if ( !rc ) // podmínka, pokud nedojde k registraci okna ukonči program
       return FALSE;
//add 3) Vytvoření okna
hMainWnd = CreateWindow("UC_GRAPH","GraphWindow",
WS_CAPTION|WS_SYSMENU|WS_MINIMIZEBOX|WS_MAXIMIZEBOX|WS_SIZEBOX, // styl
                        x, y, //pozice levého horního rohu na ploše
                        cx, cy, // velikost okna
                        NULL, // není nadřízené okno „parent"
                        NULL, // neměníme menu, použijeme to definované při registraci
                        hInst, // instance zařízení
                        NULL ); // pointer na pomocná data pro WM_CREATE
if ( !hMainWnd ) // podmínka, neplatí manipulátor okna ukonči program
                return FALSE;
                ShowWindow( hMainWnd, nCmdShow ); // zobrazí okno
                UpdateWindow( hMainWnd ); // update okna
// add 4) Smyčka zpráv
while( GetMessage( &msg, NULL, 0, 0 ) ) { // čti zprávy dokud se neobjeví WM_DESTROY
                        TranslateMessage( &msg ); // přelož zprávu
                        DispatchMessage( &msg ); // odešli zprávu callback funkci hlavního
                        okna
                }
                return msg.wParam; // ukončuje aplikaci
              }
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

2.4.3 Volání zpráv

Činnost OS Windows se opírá o systém zpráv, každý jednotlivý proces je zpracován právě přes volanou zprávu. Na mnoho zpráv systém reaguje posláním dalších zpráv. Příklad volání zpráv při vytváření hlavního okna viz Obr.1

Při volání zpráv nejprve nadefinujeme proceduru okna MainWndProc typu CALLBACK. K jednotlivým zprávám se dostaneme pomocí přepínače switch, který porovnává parametr uMsg a zavolá potřebnou zprávu pomocí její hodnoty v stupni case. Zprávy jsou zpracovány pomocí Smyčky zpráv, dokud nedojde k volání zprávy WM_DESTROY [5]. Funkce DefWindowProc na závěr vrací všechny nezpracované zprávy zpět do systému. Následuje fragment zdrojového kódu a popis jednotlivých, v našem případě použitých, zpráv:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
LRESULT CALLBACK MainWndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM
lParam ) {
    switch ( uMsg ) {
        {
        case WM_CREATE:{
                }
        break;
        case WM_ PAINT:{
                }
        break;
        case WM_ SIZE:{
                }
        break;
        case WM_DESTROY:{
                PostQuitMessage(0);
                return 0;
                }
        }
     return DefWindowProc(hWnd, message, wParam, lParam);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

WM_CREATE:

bude zaslaná callback funkci okna jako zpráva první, během provádění CreateWindow, právě ve chvíli kdy je okno vytvořeno ale ještě nedošlo k jeho zobrazení . Návratová hodnota 0 udává, že lze pokračovat v tvorbě okna, zatímco pokud je vrácena hodnota 1, fce. CreateWindow vrací hodnotu NULL a dojde ke zrušení okna.

WM_PAINT:

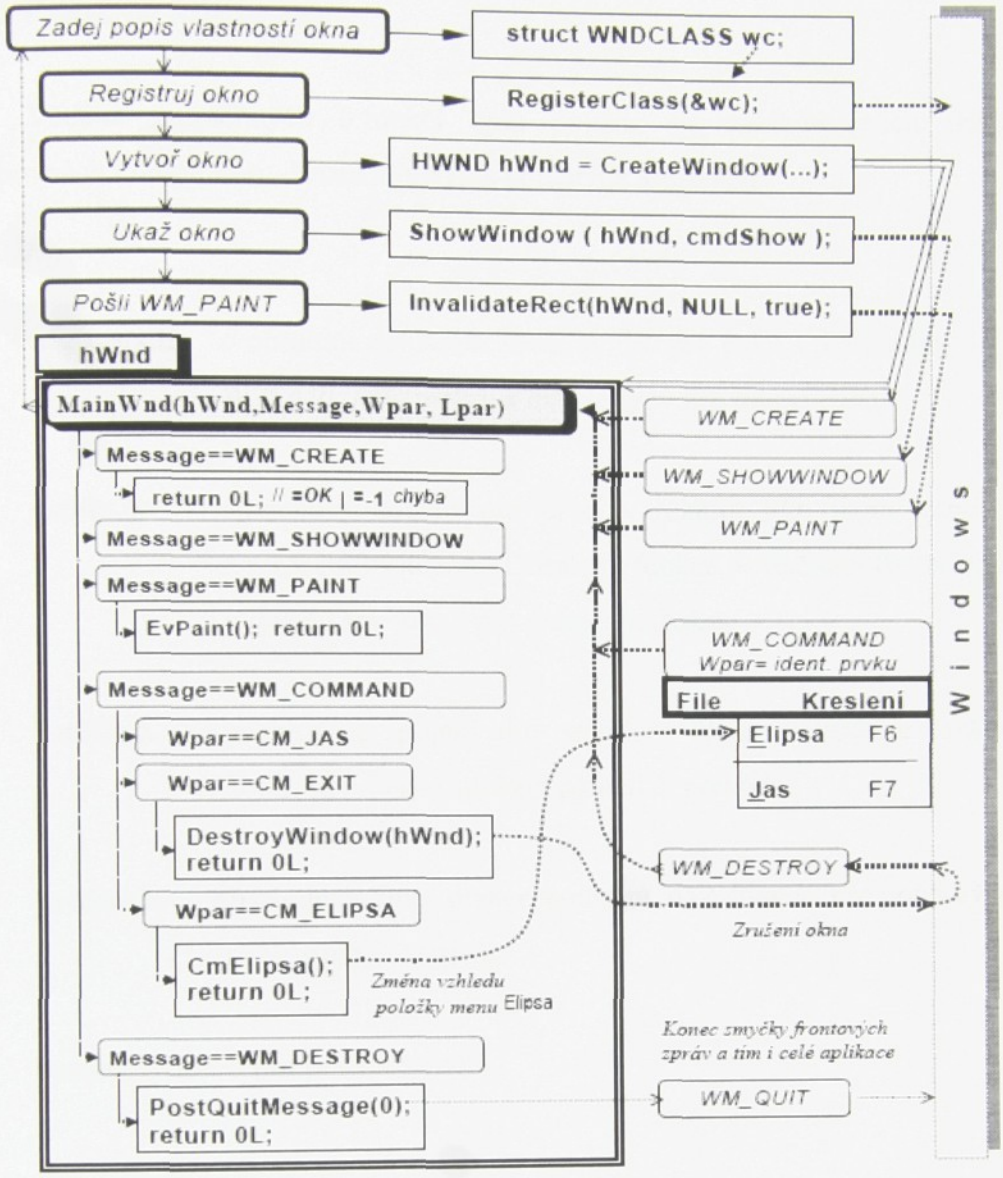
zašle OS, kdykoliv se musí znovu vykreslit obsah okna, informace o funkcích pro kreslení. Dále ji lze vyvolat z programu, třeba již zmíněnou funkcí InvalidateRect.

WM_SIZE:

slouží nám pro zjištění aktuální velikosti a pozice okna, kterou můžeme následně použít například pro roztahování grafu v závislosti na velikosti okna. Používáme v kombinaci s fcí. GetClientRect.

WM_DESTROY:

těsně před zrušením okna dostaneme volání DestroyWindow. V našem programu při volání zprávy WM_DESTROY dojde k provedení fce PostQuitMessage(0), čímž se pošle další zpráva WM_QUIT s parametrem udávajícím zadaný 0 exit kódu programu (*wParam* = 0).Zpráva WM_QUIT způsobí, že funkce GetMessage vrátí 0, čímž se ukončí smyčka frontových zpráv ve WinMain a dojde k ukončení celé aplikace.



Obr.1: Tok zpráv při vytváření okna

2.4.4 Dětská okna

V překladu **Child Windows**. Tato okna v aplikaci byla vytvořena pomocí **MDI** (multi -document interface) na místo klasické **SDI** (single- document interface). Výhodou **MDI** rozhraní je (v našem případě) možnost zobrazení více grafů ve stejném čase. Každý graf pak bude zobrazen ve vlastním okně. Okna jsou podřízena mateřskému a proto volání zpráv určených pro vykreslení grafů, popisků os atd. je zpracováno právě v rámci dětského okna. Pro každé okno je možné připravit požadované menu a tlačítka minimalizace, maximalizace a zrušení aplikace. Kód tohoto okna není uveden, protože je téměř totožný s kódem okna hlavního.

2.4.5 Použité knihovny

Většina proměnných a funkcí byla použita na základě jejich deklarace v knihovnách kompilátoru. Následuje výpis použitých knihoven:

<**windows.h**> // obsahuje deklarace všech funkcí pro práci s rozhraním WINAPI, dále makra a datové typy pro toto rozhraní

<**stdio.h**> // hlavičkový soubor pro práci v režimu I/O (input/output) tedy s vstupem/výstupem. Obsahuje funkce na načítání dat ze souboru, výpis do souboru a pro práci s bufferem

<**stdlib.h**> // pochází ze spojení slov **st**andard **lib**rary, obsahuje funkce pro kontrolu procesu, alokaci paměti či převod.

<**string.h**> // obsahuje definice maker, konstant, dále funkce. pro práci s řetězci např. strcpy či strlen.

<**memory.h**> // pro práci s pamětí

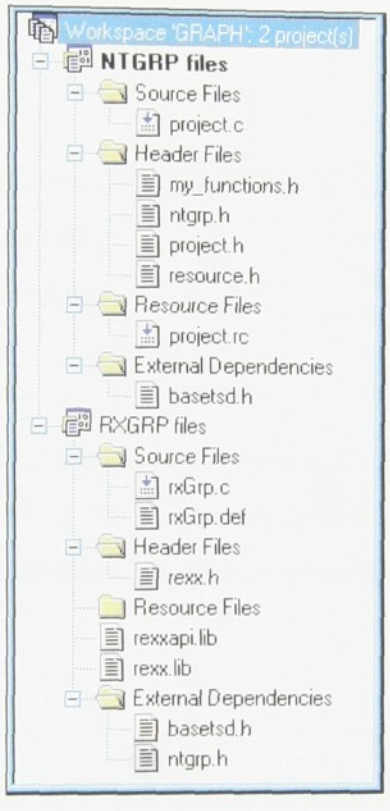
<**process.h**> // obsahuje deklarace rodiny funkcí **exec**, fungujících na principu nahrazení běžícího procesu jiným po jejich zavolání

3 Vytváření aplikace v OS Windows XP

Programovacím nástrojem se stalo vývojové prostředí Microsoft Visual Studio C++ verze 6. Důvodem byla snadná obsluha a předchozí používání na hodinách programování pod záštitou katedry KKY. Z tohoto důvodu bude popis vývoje aplikace směrován právě do tohoto prostředí.

3.1 Pracovní plocha

Pracovní plocha (Solution) obsahuje dva projekty (Obr. 2): NTGRP, RXGRP. První z nich (NTGRP) obsahuje zdrojový kód aplikace pro vykreslování grafů z naměřených dat, druhý (RXGRP) obsahuje zdrojový kód dynamické knihovny (DLL) s funkcemi pro programovací jazyk REXX pomocí kterých lze aplikaci z projektu NTGRP ovládat.



Obr. 2: Vzhled naší aplikace v programu Visual Studio

Součástí projektu NTGRP je soubor ntgrp.h. Hlavičkový soubor ntgrp.h obsahuje definice použitých datových struktur pro obě části programu, proto je přidán do obou částí. Dále jsou zde soubory my_function, obsahuje definice funkcí a datových typů, project.c, obsahuje výpis funkcí pro práci ve WINAPI, project.h s definici fce WinMain, resource.h s definicí prostředků a project.rc s prostředky a jejich editací.

Součástí projetu RXGRP, je dynamická knihovna rxGrp.c, která obsahuje deklarace funkcí pro jejich externí volání přes systém zpráv WM_USER, nedeklarovaných v souborech rxGrp.c a project.c. Soubor rxGrp.def pak obsahuje výpis funkcí ze souboru rxGrp.c. Dále pak statické knihovny rexxapi.lib a rexx.lib pro použití jazyka REXX.

3.2 Vývoj aplikace

3.2.1 Funkce projektu NTGRP

V této sekci se blíže seznámíme s funkcemi programu, které deklarovány v souboru my_functions.h, vysvětlíme si jejich princip a uvedeme příklady zdrojového kódu.

1) *AddGraph*

- parametry (int ***iCountCurves***, int ***iScaleFromX***, int ***iScaleToX***, int ***iScaleFromY***, int ***iScaleToY***)
- Obsahuje definice MDI okna.

Princip:

do proměnné ***iCountCurves*** uloží počet křivek a naalokuje matici pro načtení bodů ze souboru. Na konci inkrementuje číslo grafu a pokračuje s alokací až do splnění podmínky. Následuje volání funkce **UpdateGraph()**.

2) *LoadCurve*

- parametry (char ***pFile**, int ***iIndexGraph**, int ***iIndexCurve***)

Princip:

Otevře ***pFile*** soubory deklarované na začátku souboru **fr = fopen(pFile,"r")**, nepodaří-li se soubor otevřít či obsahuje-li neplatné hodnoty, objeví se dialogové okno s chybovou hláškou. Následuje načtení všech hodnot křivek grafu a následné přiřazení hodnot automatického měřítka výběrem minimálního a maximálního prvku křivek v obou osách. Funkce vrací hodnotu TRUE pokud se proces zdaří.

3) *PrintGraph*

- parametry (HWND ***hWnd,*** BOOL ***bUseScale***)
- **hWnd** (držadlo, rukojeť) je proměnná typu **HWND**, čili identifikátor objektu. Použijeme-li handle, systém se bude jednoznačně odkazovat na dané zařízení právě pomocí tohoto identifikátoru. Dalšími handle identifikátory jsou například **hpen** typu **HPEN** pro pero, **hbrush** typu **HBRUSH** pro štětec, či **hdc** typu **HDC** především pro vnitřní procesy funkcí.

Princip:

Slouží pro vykreslení křivek do grafu. Používáme zde fce. **BeginPaint(hWnd, &ps**), kde **ps** je proměnná struktury **PAINTSTRUCT**, pro začátek a **EndPaint(hWnd, &ps)** pro ukončení kreslení. Na počátku získáme handle hlavního okna a jeho rozměry pomocí fce. **GetClientRect(hWnd, &rt)** kde **rt** je proměnná typu **RECT**. Následuje vykreslení obdélníku a nastavení barvy pera a druhu čáry pro kreslení rastru. Pro vykreslování křivek a rastru použijeme fce. **MoveToEx** a **LineTo**, kdy dochází ke spojení jednotlivých bodů přímkami. Fcí pro vykreslení řetězce je **TextOut**, obsahující handle zařízení, zde dětského okna, a dále pozici, délku textu a proměnnou obsahující řetězec. Při vykreslení textu nesmíme zapomenut na nastavení a zrušení fontu. Můžeme přikročit k samotnému zobrazení grafu. V této části funkce je na prvním místě nastaveno měřítko, v závislosti na požadavku uživatele a sice, pokud proměnná ***bUseScale*** bude nabývat hodnoty FALSE, dojde pouze k roztažení křivky v rámci grafu, v dalším případě si fce. nastavuje měřítko z počátečních hodnot a poslední možností je využití fcí. **SetScaleX** a **SetScaleY** pro volbu měřítka uživatelem za běhu programu. Dojde k vykreslení křivek, bohužel i mimo vlastní ohraničení grafu. Pro korekci lze s výhodou užít fcí. **CreateRectRgn**, kterou vytvoříme oblast k oříznutí, a dále **SelectClipRgn**, pro výběr oblasti. Následně nastavíme popisky os a jejich formátování, fce **SetTextAlign**. Posledním kroky jsou pak zrušení všech použitých per,štětců, fontů, a nastavení původních a vykreslení obdélníku ohraničujícího křivku.

4) RemoveGraph

- parametry (HWND ***hWnd***)

Princip:

Slouží pro zrušení grafu a uvolnění naalokovaných polí.

5) SetScaleX

- parametry (int ***nGraphXX,*** int ***fromX,*** int ***toX***)

Princip:

Jedná se o fci. pro změnu měřítka uživatelem v ose X za běhu programu. Po zavolání fce. a načtení čísla grafu do proměnné ***nGraphXX*** a dále počátečního a koncového bodu měřítka do proměnných ***fromX, toX***, dochází k překreslení grafu v rámci fce. **PrintGraph** a nastavení nových hodnot na ose X.

6) SetScaleY

- obdoba fce. **SetScaleX** pro osu Y.

7) SetTitleX

- parametry (int ***nGraph,*** char ***name[255]***)

Princip:

Po zavolání této funkce dojde k načtení čísla grafu do proměnné ***nGraph*** a řetězce do proměnné ***name***. Číslo grafu je dále přiřazeno do globální proměnné. Načtený řetězec nelze pouze nakopírovat, proto využijeme standardně definované fce. char **strcpy** (char * ***destination,*** const char * ***source***) s dvěma parametry a sice výsledné proměnné a výchozí. Obě globální proměnné jsou zpracovány fcí. **PrintGraph**.

8) SetTitleY

- obdoba fce. **SetTitleX** pro osu Y.

9) SetWindowTitle

- parametry (HWND ***hwnd,*** CHAR * ***str***)

Princip:

Je založena na použití fce. pro změnu titulku hlavního okna: BOOL **SetWindowText**(HWND ***hwnd***, LPCTSTR ***str***). Řetězec je načten po zavolání naší fce. a dále zpracován.

9) UpdateGraph

- neobsahuje žádné parametry
- používáme ji pro detekci polohy a automatickou změnu velikosti dětských oken a grafů

Princip:

Pomocí fce. **GetClientRect**(***hMainWnd, &rect***) jsou získány rozměry hlavního okna a uloženy do proměnné ***rect***. Následuje použití fce. pro nastavení pozice dětských oken **SetWindowPos**(***hClientWnd, NULL, 0, 0, rect.right - rect.left, rect.bottom - rect.top,SWP_NOZORDER***).

3.2.2 Funkce projektu RXGRP

Tyto funkce se nacházejí ve zdrojovém souboru rxGrp.c. Stejně jako u původní aplikace jsou na začátku souboru definice a odkazy na knihovnu REXXu. Novinkou je deklarace zpráv volaných v kódu aplikace a deklarace funkcí pro použití v souboru *.rex (Obr. 3).

```
#define  INVALID_ROUTINE 40
#define  VALID_ROUTINE    0

#define BUILDRXSTRING( t, s )
{ strcpy( ( t ) -> strptr,( s ) );
( t ) -> strlength = strlen( ( s ) );
 }


#define UM_SETTITLE  WM_USER + 1
#define UM_SETXLABEL WM_USER + 2
#define UM_SETYLABEL WM_USER + 3
#define UM_SETXSCALE WM_USER + 4
#define UM_SETYSCALE WM_USER + 5
#define UM_ADDGRAPH WM_USER + 6
#define UM_ADDCURVE WM_USER + 7

static PSZ  RxFncTable[] = {

   "grpLoadFuncs",
   "grpDropFuncs",
   "grpInitializeGraph",
   "grpTerminateGraph",
   "grpSetGraphTitle",
   "grpSetXLabel",
   "grpSetYLabel",
   "grpSetXScale",
   "grpSetYScale",

};
```

Obr. 3: deklarace funkcí souboru rxGrp.c

Definice:

INVALID_ROUTINE, VALID_ROUTINE

- konstanty nabývající hodnot FALSE či TRUE. Slouží například pro posouzení správného načtení ze souboru

BUILDSTRING

- makro pro vytvoření řetězce, první položkou pole znaků, druhou délka pole

UM_SETTITLE WM_USER + 1

UM_ADDCURVE WM_USER + 7

- uživatelské zprávy použité pro volání exportovaných fcí. z knihoven REXXu a jejich definice

static **PSZ RxFncTable[]**

- zde definujeme naše funkce pro práci s měřítkem, popisky,načtení grafu, atd.

V této části došlo pouze k deklaraci funkcí, konstant, uživatelských zpráv, přiřazení hlavičkových souborů a knihoven pro práci s REXXem.

V části druhé deklarujeme výše zmíněné funkce a jejich export do prostředí REXXu. K tomu nám slouží funkce __declspec(dllexport).

Výpis funkcí:

1) ***grpLoadFuncs***

- parametry (PCHAR ***name***, ULONG ***numargs***, RXSTRING ***args[]***, PCHAR ***queuename***, PRXSTRING ***retstr***)
- význam parametrů je následující:

name – název funkce

numargs – obsahuje počet parametrů volané funkce
(pro funkci nastav titulek potřebujeme argumenty dva handle okna a řetězec)

args[] – pozice uložení jednotlivých argumentů
(pozice 0 - handle)

queuename – pořadové jméno

retstr – proměnná obsahující návratovou hodnotu

Princip:

Tato funkce je stěžejní, používáme ji k načtení všech ostatních, námi deklarovaných fcí. Nemusíme tak načítat každou zvlášť.

2) ***grpDropFuncs***

– parametry jsou shodné s předchozí i s následujícími funkcemi, nebudeme je tedy dále uvádět

Princip:

Po postupném zpracování všech volaných funkcí v souboru **test.rex** se postará o jejich ukončení a uvolnění paměti. K tomu slouží fce. **RexxDeregisterFunction** (**RxFncTable**[*j*]), kde proměnná ***j*** typu int je počet volaných funkcí.

3) ***grpSetGraphTitle***

Princip:

Nejprve pomocí klasické podmínky používané i v ostatních funkcích if (***numargs*** != ***1***)zjistíme, zda jsou načteny všechny parametry, zde načítáme pouze řetězec, proměnná ***numargs*** má tedy parametr pouze jeden. Pokud neplatí podmínka funkce vrací konstantu **INVALID_ROUTINE**, opačném případě pokračuje tělem funkce a po jejím ukončení vrací konstantu **INVALID_ROUTINE**. V těle fce. najdeme přiřazení hodnot proměnným struktury TITLEDATA pomocí klasické fce. **sprintf(** ***td.title***, "***%s***", (CHAR *)(***args[0].strptr***)), kde **td.title** je proměnná do které ukládáme řetězec typu char s pointrem na první prvek pole **args[]**. K zaslání zprávy do jiného procesu existuje fce. **SendMessage**(***hwnd***, **WM_COPYDATA**, (WPARAM)***NULL***, (LPARAM)&***cds***), která posílá pouze řetězce a nelze použít na číselné hodnoty. Prvním jejím parametrem je handle klientského okna, druhým volaná zpráva, a zbývající dvě proměnné přenesou do aplikace informace o velikosti a typu proměnné.

4) *grpSetXLabel*

Princip:

Pracuje na velmi podobném principu jako fce. ***grpSetGraphTitle***, liší se pouze počtem argumentů. Zde jsou potřeba parametry dva a sice první z nich obsahuje číslo grafu a druhý opět řetězec znaků typu char. Protože se zjevně jedná o číslo použijeme fci. **atoi**, která nám vrací velikost proměnné, v tomto případě ve tvaru ***ld.n*** = **atoi**(***args[0].strptr***). ***ld.n*** je potom proměnná, kterou voláme ve zprávě **WM_SETXLABEL**, a je do ní uložena velikost prvního prvku pole, což je právě ono zmiňované číslo grafu.

5) *grpSetYLabel*

– obdoba fce. ***grpSetXLabel*** pro osu Y.

6) *grpSetXScale*

Princip:

Tato fce. volá tří parametry, číslo grafu, počáteční a koncový bod měřítka. Do první proměnné ***sd.n*** struktury SCALEDATA je uloženo právě číslo grafu s využitím fce. **atoi**. A proměnné ***sd.min*** a ***sd.max*** stejné struktury obsahují body měřítek. Jelikož však chceme volit měřitko s desetinným dělením, nelze použít fce. jiná než **atof**, která převede parametr na číslo desetinné. Př. ***sd.min*** = **atof**(***args[1].strptr***)

7) *grpSetYScale*

– ekvivalent fce. ***grpSetXScale*** pro osu Y.

8) *grpAddGraph*

Princip:

Na stejném principu jako dvě předchozí fce. pracuje i tato pro přidávání nového grafu. Načítáme zde pět hodnot, a sice chronologicky, číslo grafu, počáteční hodnota minima osy X, počáteční hodnota maxima osy X, počáteční hodnota minima osy Y a na závěr počáteční hodnota maxima osy Y. Po zavolání této fce. v souboru **test.rex** dochází vždy k přidání nového grafu, pokud nechceme tento přidat je nutné fci. včas odstranit z prostoru pro vkládání.

9) *grpRemoveGraph*

Princip:

Pracuji na bázi uvolnění všech bodů, jejich spojnic, měřítek a rastru grafu. Po zavolání této funkce dojde k uvolnění pomocí funkce free() a funkce DestroyWindow následně zruší okno. Funkce načte číslo grafu do proměnné sd.no za použití funkce atoi(), které je použito v původní funkci RemoveGraph() ve tvaru RemoveGraph(graphs[sd.no-1].hWnd). Dochází ke zrušení grafu a překreslení okna.

10) *grpSetData*

Princip:

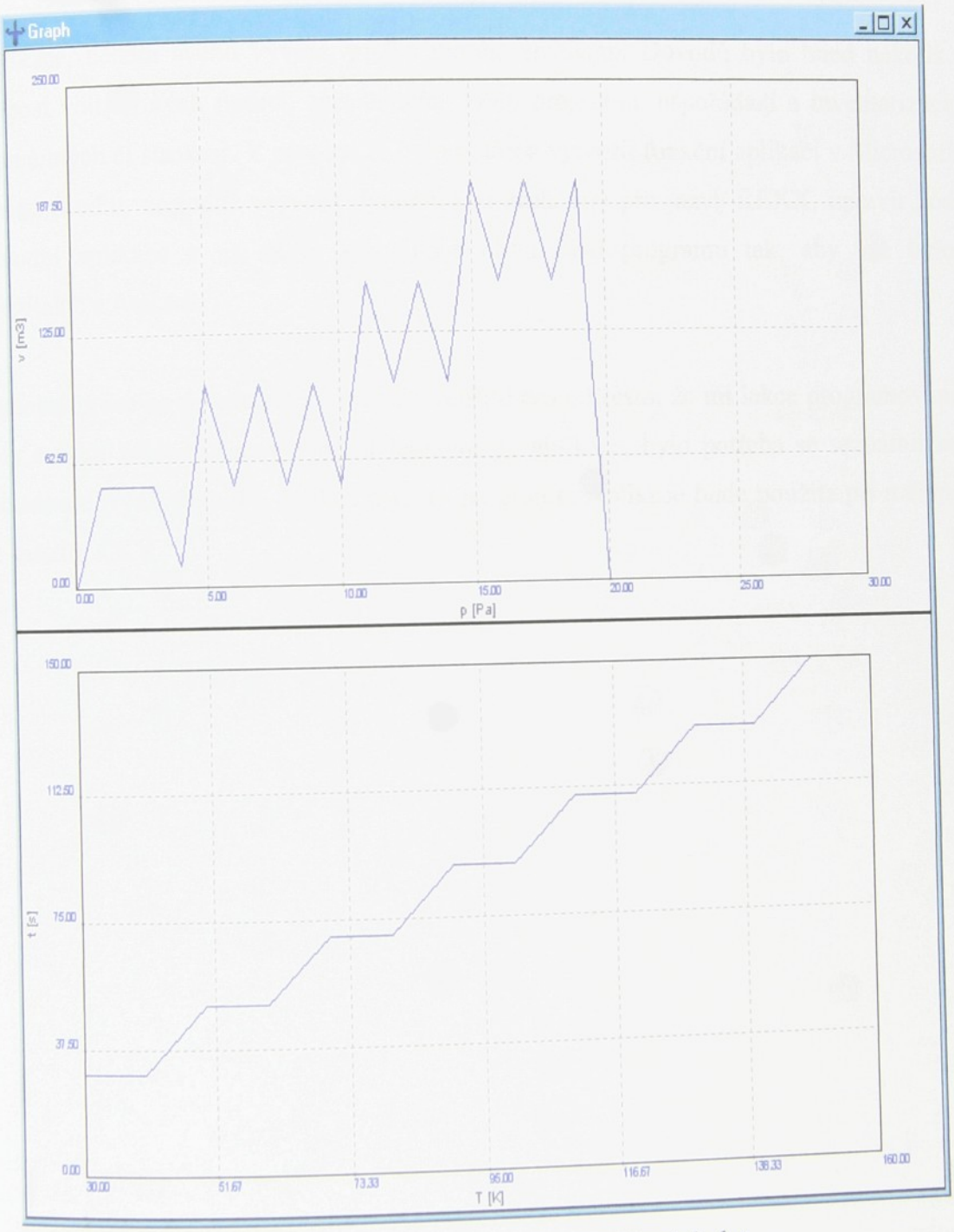
Pomocí cyklu for načteme data do naalokovaného pole, program s nimi pracuje pouze jako s řetězcem, proto využijeme funkce atof a převedeme je tak na číselné hodnoty. Následuje volání funkce AddCurve s parametry pole dat a vykreslení jedné křivky. Pokračujeme načtením a vykreslením křivek až do poslední z nich.

Následující fragment zdrojového kódu, napsaném v jazyku REXX, demonstruje způsob užití vyvinuté dynamické knihovny RXGRP.

```
/* */
rc = RxFuncAdd("grpLoadFuncs","RXGRP","grpLoadFuncs")

/* načítá všechny funkce */
rc = grpLoadFuncs()

/* ************************************************ */
/* ** PROSTOR PRO VKLADANI VLASTNICH FUNKCI** */
/* ************************************************ */
rc = grpSetGraphTitle( "RETEZEC" )        /* název okna */

rc = grpSetXLabel( 1, "p [Pa]" )      /* titulek osy X, grafu 1 */
rc = grpSetYLabel( 1, "v [m3]" )      /* titulek osy Y, grafu 1 */

rc = grpSetXScale( 1, 10, 30 )        /* měřítko osy X, grafu 1 */
rc = grpSetYScale( 1, 50, 100 )       /* měřítko osy Y, grafu 1 */


/* ************************************************ */
/* *************** KONEC PROSTORU *************** */
/* ************************************************ */

/* ukončuje, uzavírá všechny funkce*/
rc = grpDropFuncs()
```

4 Závěr

V bakalářské práci byla vytvořena funkční aplikace pro vykreslování grafů ze souboru, včetně možnosti volitelného měřítka, popisků os, popisku hlavního okna, přidáváni grafů, viz následující Obr. 4.



Obr.4: Aplikace pro vizualizaci měřených dat

Aplikace byla vytvořena ve vývojovém prostředí Microsoft Visual Studio verze 6, v programovacím jazyku C.

Pro ovládání uvedené aplikace byla vytvořena dynamická knihovna funkcí pro programovací jazyk REXX. To znamená, že z libovolné REXX aplikace lze nastavovat parametry grafu, např. měřítka, zobrazovaná data.

Aplikace, během svého vývoje, prošla mnoha změnami. Důvodů bylo hned několik, nutnost volby jiných funkcí, zpřehlednění kódu programu, uspořádání a inventarizace proměnných či struktur. V první řadě bylo potřeba vytvořit funkční aplikaci v Microsoft Visual Studiu, následně vytvořit dynamickou knihovnu pro jazyk REXX, upravit kód původní aplikace a na závěr kombinace obou částí programu tak, aby vše bylo použitelné a funkční.

Samotná práce na programu mi zabrala většinu času. Přesto, že mi lekce programování daly solidní základy pro vytvoření jednoduché aplikace, bylo potřeba se seznámit se spoustou nových funkcí a aplikovat je do programu. Aplikace bude použita při měření na katedře KKY.

5 Použitá literatura a internetové zdroje

[1] HEROUT, P.: *Učebnice jazyka C - 1.díl*, Nakladatelství Kopp, 2004

[2] HEROUT, P.: *Učebnice jazyka C - 2.díl*, Nakladatelství Kopp, 2004

[3] Kolektiv autorů: *Programování v jazyku C*, JZD AK Slušovice, 1987

[4] ŠUSTA, R.: *Programování pro řízení ve Windows*, ČVUT-FEL, 1999

[5] Seriál WIN32API <http://www.programujte.com/>

[6] Učíme se WINAPI <http://www.builder.cz/art/cpp/winapi1.html>

[7] Základy WINAPI < http://tamnekde.unas.cz/data/prg/prg_g1/>

[8] Seriál o C++ pro úplné začátečníky < http://www.zive.cz/h/Programovani/>

[9] Otevřená encyklopedie < http://cs.wikipedia.org/>

[10] < http://mozek.cz/info/jazyk-c>

[11] < http://www-306.ibm.com/software/awdtools/rexx/library/rexxhist.html>

[12] Jazyk C/C++ http://www.sallyx.org/sally/c/c11.php

[13] MSDN <http://msdn2.microsoft.com/en-us/library/default.aspx>

[14] <http://www.geocities.com/siliconvalley/garage/3323/cz_rexxpage.html >

[15] ZÁBRODSKÝ, V.: *Album algoritmů a technik*, Nico Mak Computing, 2003