1 Procedurální programování a strukturované programování

Charakteristické pro procedurální programování je organizace programu, který řeší daný problém, do bloků (procedur, funkcí, subrutin). Původně jednolitý, a nepřehledný program, je rozdělení úlohy na dílčí podúlohy, které jsou implementovány v jednotlivých funkcích. Každý procedurální programovací jazyk má vnitřně implementované konstrukty pro proramové cykly (for cykly, while cykly). Navíc je zpravidla přidána možnost sestavení nových datových struktur z vnitřně definovaných.

2 Datové typy v jazyce C

Základní jednotou informace je bit, ta nabývá dvou hodnot 0 nebo 1. Prakticky si realizaci bitu můžeme představit jako přepínač, kdy poloha on odpovídá 1 a poloha off odpovídá 0. V paměti počítače jsou bity organizovány do větší struktury obsahující 8 bitů. Tato jednotka je pojmenována byte.

Table 1: Datové typy v C

jmého datového typu	velikost v bytech (B)	obsah
char	1	znak
int	4	celé číslo
float	4	číslo s plovoucí desetinou čárkou
double	8	číslo s plovoucí desetinou čárkou

3 Odvozené datové typy

Pomocí konstrukce typedef

```
struct{<structure body>}<NAME>
```

můžeme odvodit vlastní typy které obsahují více položek. Například definujme strukturu VECTOR.

```
typedef struct{float x,y,z; }VECTOR;}
```

V programu pak deklarujete, např., proměnnou v jako typ VECOR následně

```
VECTOR v;
```

K jednotlivým prvkům struktury pak přistupujet pře operátor tečka, tj., ".". Podívejte se na následující příklad

```
p.x=1.0;
p.y=2.0;
p.z=7.0;
```

4 Pole

Pro potřeby automatického zpracování více dat stejného datového typu se dobře hodí pole. Pole je od ostatních proměnných odlišuje hranatými závorkami za jménem proměnné. Například deklarujeme pole A o 10 prvcích datového typu int,

```
int A[10];
```

Jednotlivé prvky pole jsou od sebe odlišeny intexem od 0 do 9, tj., prvky výše deklarovaného pole A jsou

```
A[0], A[1], A[2], ... , A[9].
```

Takto jsem si ukázali deklaraci jednozorměrného pole. Pro případ dvourozměrného pole je postup následující,

```
int B[2][10];
```

Tímto příkladem jsme deklarovali pole B které má 2 řádky a 10 sloupců. U vícerozměrných polí postupujeme analogicky.

5 Funkce main

Ve svém programu můžete definovat "bezpočet" funkcí. Mezi nimi však musí být jedna vyjímečná, která je volána systémem jako první, je to funkce `main`. Každá funkce se skládá z hlavičky obsahující vrácený datový typ, název funkce a v kulatých závorkách argumenty funkce. V případě funkce main budeme používat její hlavičku ve tvaru

```
int main(int c, char *argv[])
```

kde je vidět, že vrací hodnotu typu int, prvním argumentem je počet argumentů na příkazové řádce, a jejím druhým argumentem je pole řetězců obsahující hodnoty argumentů na příkazové řádce. Uveďmě jednoduchý příklad programu v C.

6 Hello world

V této sekci uvádím sérii variací na příklad HelloWorld. Začněme tou nejjednodušší první variantou

```
#include<stdio.h>

int main(int c, char *argv[])
{
        printf("Hello world\n");
        return 0;
}
```

Na prvním řádku je tzv. direktiva preprocesoru pro vkládání jiného souboru. V tomto případě je to soubor `stdio.h`, který obsahuje hlavičky ke standartním vstupně/výstupním (I/0) fukcím. Na druhém řádku je hlavička hlavní funkce, funkce main. Tělo funkce je uvozeno kroucenými závorkami. V těle funkce jsou dva řádky. Na prvním řádku voláme funkci `printf`jejímž argumentem je řetězec, který má být vytištěn na standartní výstup (většinou na obrazovku). Všimněte si, řádek je ujončen středníkem(";"). Středník vždy ukončuje řádek. Na druhém řádku je příkaz return, kterým systém informujeme o stavu ukončeného programu (zda proběhl vpořádku, pokud ne tak jaká nastala chyba).

V dalším příkladu si ukážeme jak vytisknou na obrazovku počet argumenů na řádce taky jednotlivé argumenty.

```
#include<stdio.h>

int main(int c, char *argv[])
{
        int i;
        printf("Hello world\n");
        for(i=0;i<c;i++)
```

```
        {
                printf("argv[%d]=%s\n",i,argv[i]);
        }
        return 0;
}
```

V tomto příkladu přibylo několik nových řádků. Nejprve si všimněte deklarace proměnné (`int  i;`). Tímto řádkem jsme v paměti počítače staticky alokovali 4 byty pro celočíselnou proměnou `i`,která slouží k indexaci pole `argv`.Za voláním příkazu `printf`přibyl cyklus `for`, ve které se bude v každém kroku cyklu zvyšovat hodnota uložená v `i`o jedničku (`i++`) a to postupné od 0 až po hodnotu uloženou v proměnné `c`. V každém kroku cyklu je volána funkce printf, která vytiskne na obrazovku hodnotu indexu `i`a hodnotu argumentu `argv[i]`.

7 Řešení kvadratická rovnice

Když zapíšeme obecnou kvadratickou rovnici ve tvaru

$$ax^2 + bx + c = 0 \tag{1}$$

tak její řešení hledáme ve tvaru

$$x_1 = \frac{-b + \sqrt{D}}{2a} \tag{2}$$

$$x_2 = \frac{-b - \sqrt{D}}{2a} \tag{3}$$

kde je D diskriminant rovnice, jehož hodnotu určíme z výrazu

$$D = b^2 - 4ac \tag{4}$$

Před samotným programováním musíme provést analýzu úlohy. Jaké hodnoty mohou nabývat parametry a,b,c. Jestli D může být libovolné číslo, atp. V našem případě musí být $a! = 0$ a $D > 0$(pokud hledáme řešení pouze v prostoru reálných čísel). Výsledný program by mohl vypadat takto

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int main(int nc, char *argv[])
{
```

```
        float a,b,c,D,x1,x2;

        if(nc==4)
        {
                a=atof(argv[1]);
                b=atof(argv[2]);
                c=atof(argv[3]);

                if(fabs(a)>1.0e-8)
                {
                        D=b*b-4.0*a*c;
                        if(D>0.0)
                        {
                                x1=(-b+sqrt(D))/(2.0*a);
                                x2=(-b-sqrt(D))/(2.0*a);

                                printf("x1=%f  x2=%f\n",x1,x2);
                        }else{
                                printf("reseni neni v prostoru realnych cisel\n");
                                return -3;
                        }
                }else{
                        printf("neni kvadraticka rovnice\n");
                        return -2;
                }
        }else{
                printf("nedostatek argumentu\n");
                return -1;
        }

        return 0;
}
```

.... další text bude v brzku následovat :-) ....