

Koncepce programu

7. cvičení

Jiří Zacpal

KMI/UP2 – Úvod do programování 2

2. úkol

Napište funkce:

```
long double prumer(int pocet, ...)  
long double median(int pocet, ...)  
long double rozptyl(int pocet, ...)
```

které pro proměnné typu `double`, jejichž počet je zadán proměnnou `pocet` vrátí aritmetický průměr, medián a rozptyl zadaných proměnných.

Tyto funkce použijte ve funkci

```
void statistika_roku(int pocet, pole_funkci* fce, double *udaje),
```

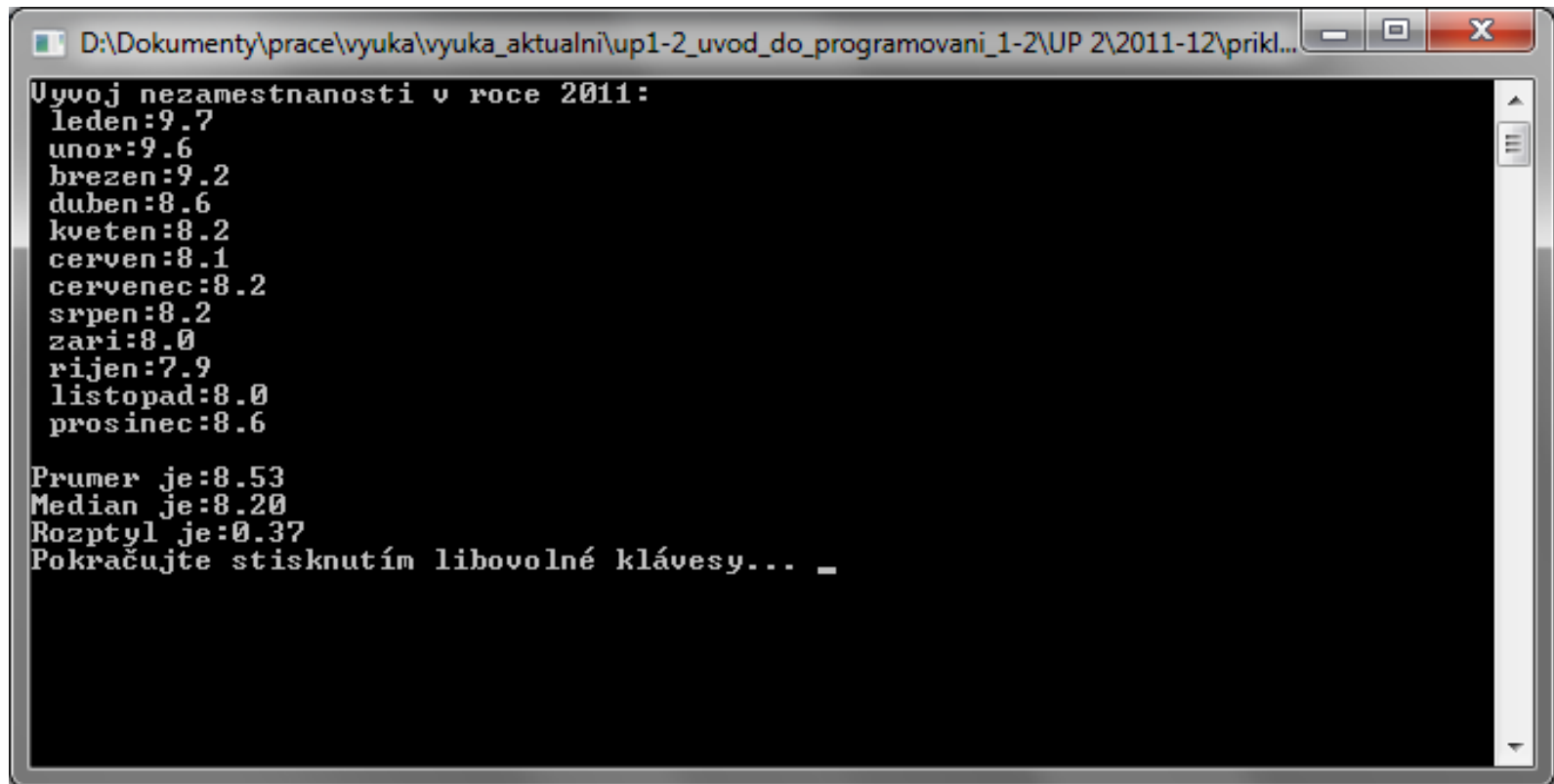
která pro údaje za jednotlivé měsíce roku uložených v poli `udaje` spočítá jednotlivé funkce, jejichž počet je uložen v proměnné `pocet`.

Typ `pole_funkci` je tato struktura:

```
typedef struct pole_funkci  
{  
    long double(*p)(int pocet,...);  
    char typ; //identifikace funkce (prumer, median, rozptyl)  
};
```

2. úkol

Příklad spuštěného programu:



```
D:\Dokumenty\prace\vyuka\vyuka_aktualni\up1-2_uvod_do_programovani_1-2\UP 2\2011-12\prikl...
Uvoj nezamestnanosti v roce 2011:
leden:9.7
unor:9.6
brezen:9.2
duben:8.6
kveten:8.2
cerven:8.1
cervenec:8.2
srpen:8.2
zari:8.0
rijen:7.9
listopad:8.0
prosinec:8.6

Prumer je:8.53
Median je:8.20
Rozptyl je:0.37
Pokracujte stisknutím libovolné klávesy... _
```

Parametry a návratová hodnota funkce main

- již víme, že main je funkce
- dá se říci, že funkce main „je volána“ přímo operačním systémem, tudíž ji od operačního systému mohou být předávány parametry a její návratová hodnota může být operačním systémem dále zpracována
- funkce main může mít žádný nebo dva formální parametry, které jsou z historických důvodů pojmenovávány `argc` a `argv`

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

- návratová hodnota funkce main by měla být typu `int`, způsob jejího zpracování však není v ANSI C nijak specifikován (každý OS ji může zpracovávat jinak)

Předání parametrů funkci main 1/2

- pokud spouštíme program z příkazového řádku (případně skriptem), lze za jménem spustitelného souboru uvést další parametry
`./xpm2eps -f *.xpm -o *.eps -by 10`
`xpm2eps.exe -f *.xpm -o *.eps -by 10`
- do parametru argc se při spuštění programu načte počet textových řetězců (oddělených mezerou) a do pole argv se načtou tyto řetězce
- ve funkci main lze hodnoty parametrů zadaných při spuštění do příkazové řádky načíst z pole argv
- měnit hodnoty proměnných argc a argv ve funkci main se nedoporučuje

Předání parametrů funkci main 2/2

- pozor, v `argv[0]` je uložen text odpovídající spouštěnému souboru (např. "xpm2eps.exe"), vlastní parametry tedy následují až na dalších indexech v poli `argv`, hodnota `argc` odpovídá počtu textových řetězců v poli `argv`
- pokud chci předat jako parametr text s mezerami, pak je nutné tento text uzavřít do úvozovek

```
pokus.exe Ahoj "jak, se" mas
```

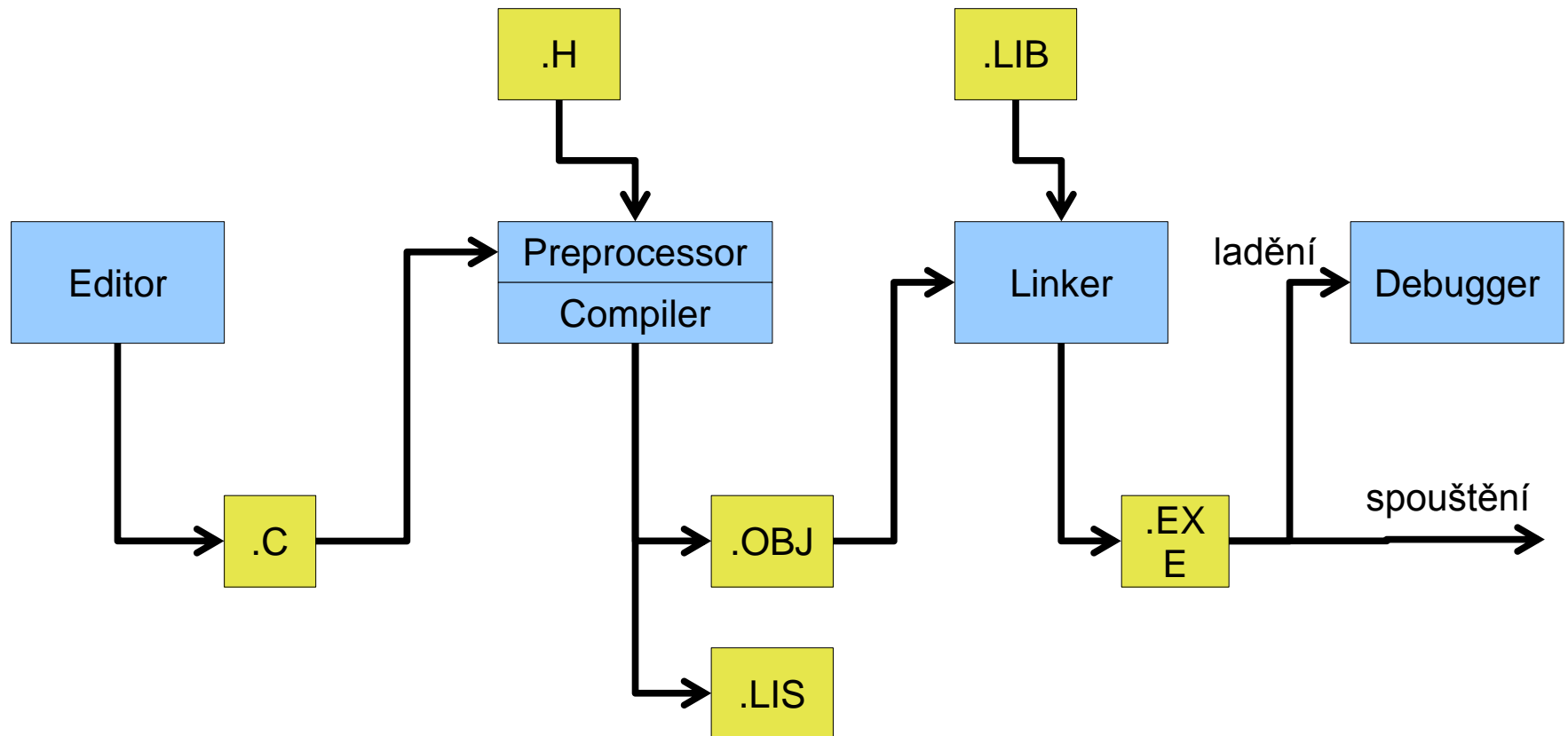
- pro konverzi textového řetězce na číslo lze použít funkci `atoi` nebo `atof`

```
int pocet = atoi(argv[1]);  
double cislo = atof(argv[2]);
```

Koncepce programu

- při vytváření větších programů (třeba i týmem více programátorů) je nutné rozdělit program do většího množství zdrojových souborů
- každý soubor by měl zajišťovat určitou „částečně samostatnou“ funkcionalitu programu (např. vstup a výstup dat, kontrola dat, část výpočtu)
- nelze se ovšem vyhnout volání funkcí z jiných souborů; někdy je potřeba také sdílení globálních proměnných, či definic typů mezi soubory
- na druhou stranu je nutné zabránit nežádoucímu sdílení identifikátorů ve chvíli, kdy náhodou programátoři zvolí pro své funkce či proměnné stejná jména

Vznik programu (schéma)



Oddělený překlad zdrojových souborů

- umožňuje relativně samostatně spravovat jednotlivé části větších programů
- zdrojové soubory jsou samostatně překládány, vznikají tzv. OBJ soubory, které jsou následně spojeny do spustitelného souboru
- při změně v jednom zdrojovém souboru není nutné překládat všechny soubory, stačí zkompilevat změněný a spojit se staršími OBJ soubory ostatních částí programu
- pokud na programu pracuje více lidí, nemusí sdílet zdrojové soubory, ale jen OBJ soubory (nehrozí nežádoucí změny cizích částí programu)

Oddělený překlad prakticky

- ve vývojových prostředích (Visual Studio, Xcode, CodeBlocks, ...) se překlad všech souborů i jejich spojení provádí většinou automaticky jediným kliknutím
- soubory jsou organizovány do projektů, některá vývojová prostředí dokonce neumoží kompilovat soubor, pokud není přiřazený k žádnému projektu
- v prostředí Linuxu (UNIXu) se pro překlad větších programů často používá utilita **make**

Hlavičkové soubory 1/2

- již při překladu jednotlivých souborů jsou potřeba informace o funkcích (identifikátory, počet a typy parametrů, typ návratové hodnoty), které se z daného souboru volají, přestože jsou definovány v jiném zdrojovém souboru
(případně informace o sdílených proměnných, typech)
- k předání těchto informací nám slouží tzv. *hlavičkové soubory*, které obsahují *hlavičky* funkcí = deklarace funkcí s uvedenými typy parametrů
(případně další informace)
- tyto soubory se následně přidávají pomocí direktivy `include` do všech souborů, které používají funkce z jim odpovídajícího zdrojového souboru

Hlavičkové soubory 2/2

- hlavičkové soubory mívají standardně příponu .h, jejich jméno obvykle odpovídá zdrojovému souboru (přípona .c), který popisují
- u větších projektů je obtížnější se vyznat v závislostech mezi soubory, proto se používá následující struktura hlavičkových souborů, která zabraňuje vícenásobnému vložení téhož souboru

```
#ifndef JMENO_H
#define JMENO_H
...
#endif
```
- pokud bychom vkládali pomocí `include` přímo zdrojové soubory a ne pouze hlavičkové soubory, ztratili bychom výhody odděleného překladu

Sdílení proměnných a funkcí

- pokud je potřeba v některé části zdrojového kódu přistupovat ke globální proměnné či funkci definované v jiném souboru, měla by zde být deklarována s modifikátorem `extern`

```
extern int pocet;  
extern double fce(double);
```

- sdílená proměnná / funkce je tedy definována pouze v jediném souboru, v dalších částech programu je pouze deklarována jako externí (s modifikátorem `extern`)
- deklarace sdílených proměnných a funkcí tvoří často hlavičkové soubory

Obrana proti nežádoucímu sdílení

- pokud na programu pracuje více programátorů (každý na jemu přidělených souborech), může se stát, že náhodou zvolí stejné identifikátory
- pokud chceme tomuto možnému nežádoucímu sdílení zabránit, je rozumné označit všechny nesdílené globální identifikátory modifikátorem `static`
`static int pocet = 12;`
`static double fce(double x) { ... }`
- při použití `static` „nepomůže“ ani modifikátor `extern` v souboru, odkud bychom chtěli k dané proměnné přistupovat

Jak si udržet pořádek v programu

- větší program je dobré rozdělit do několika částečně samostatných částí – modulů
- modul bývá většinou tvořen jedním hlavičkovým a jedním zdrojovým souborem
- používejte co nejméně sdílených proměnných (nejlépe žádné) a funkcí (ty už nějaké většinou budou)
- vše, co nemá být sdílené, označte pro jistotu jako `static`
- funkční prototypy sdílených funkcí a deklarace sdílených proměnných opatřené modifikátorem `extern` umístíme do hlavičkového souboru modulu
- hlavičkové soubory přidáváme pomocí `include` do dalších modulů, nikdy „neinkludujeme“ přímo zdrojové soubory

Doporučený obsah .c souboru 1/2

1. dokumentační část – komentář
 - jméno souboru, verze
 - stručný popis modulu
 - autor, datum, ...
2. všechny potřebné `include`
 - nejprve všechny systémové hlavičkové soubory
 - pak vlastní hlavičkové soubory
3. definice sdílených globálních proměnných
 - jejich `extern` deklarace dáme do hlavičkového souboru
4. lokální `define`
 - definice konstant a maker

Doporučený obsah .c souboru 2/2

5. definice lokálních typů

- zásadně pomocí `typedef`

6. statické globální proměnné

- snažíme se omezit na co nejmenší počet

7. úplné funkční prototypy lokálních funkcí

8. definice funkcí

- funkce main (pokud se v souboru vyskytuje)
- sdílené funkce
- lokální funkce označené jako `static`

Doporučený obsah .h souboru

1. dokumentační část
 - jméno souboru, verze
 - stručný popis modulu
 - autor, datum, ...
2. podmíněný překlad proti opakovanému „inkludování“
3. případné direktivy `include` (pokud je to nezbytné)
4. definice sdílených symbolických konstant
5. definice sdílených maker s parametry
6. definice sdílených typů (zásadně pomocí `typedef`)
7. deklarace sdílených proměnných (s mod. `extern`)
8. deklarace sdílených funkcí (s modifikátorem `extern`)

Úkol

Napište v jazyku C program pro výpočet objemu a povrchu válce, pravidelného trojbokého, čtyřbokého a šestibokého hranolu. Parametry výpočtu by mělo být možné předávat programu při spuštění z příkazové řádky. Zdrojový kód programu by měl být rozdělen do 2 modulů. Modul hlavní funkce (soubor main.c) bude zajišťovat zpracování a případně načtení chybějící parametrů výpočtu, budou z něj volány funkce zajišťující vlastní výpočet a vypisovány vypočítané hodnoty na obrazovku. Druhý modul (soubory vypocet.h a vypocet.c) pak bude zajišťovat veškeré požadované výpočty. Při řešení úlohy dbejte všech zásad zmíněných na přednášce.

Příklad použití:

```
objemy_a_povrchy.exe 0 1.2 2.4 (OS Windows)  
./objemy_a_povrchy 0 1.2 2.4 (OS Linux)
```

Příklad výstupu:

```
Valec s vyskou 1.2 a polomerem podstavy 2.4  
ma povrch 54.2592 a objem 21.7037.
```

Úkol – řešení

```
#include<stdio.h>
#include<stdlib.h>
#include "vypocet.h"

int main(int argc, char* argv[])
{
    int typ= atoi(argv[1]);
    double vyska=atof(argv[2]);
    double strana=atof(argv[3]);

    switch (typ)
    {
        case 0: printf("Valec s vyskou %1.1f a polomerem podstavy %1.1f ma povrch %2.4f a objem %2.4f\n",vyska, strana, povrch_valce(vyska, strana), objem_valce(vyska, strana));break;
        case 3: printf("Trojboky hranol s vyskou %1.1f a delkou strany podstavy %1.1f ma povrch %2.4f a objem %2.4f\n",vyska, strana, povrch_3(vyska, strana), objem_3(vyska, strana));break;
        case 4: printf("Ctyrboky hranol s vyskou %1.1f a delkou strany podstavy %1.1f ma povrch %2.4f a objem %2.4f\n",vyska, strana, povrch_4(vyska, strana), objem_4(vyska, strana));break;
        case 6: printf("Sestiboky hranol s vyskou %1.1f a delkou strany podstavy %1.1f ma povrch %2.4f a objem %2.4f\n",vyska, strana, povrch_6(vyska, strana), objem_6(vyska, strana));break;
    }
    system("PAUSE");
    return 0;
}
```

Úkol – řešení (vypocet.h)

```
//vypocet.h, verze 1.0, Jiri Zacpal, 26.3.2013  
//
```

```
#ifndef VYPOCET  
#define VYPOCET
```

```
extern double povrch_valce(double, double);  
extern double objem_valce(double, double);  
extern double povrch_3(double, double);  
extern double objem_3(double, double);  
extern double povrch_4(double, double);  
extern double objem_4(double, double);  
extern double povrch_6(double, double);  
extern double objem_6(double, double);
```

```
#endif
```

Úkol – řešení (vypocet.cpp)

```
#include<math.h>
#define PI 3.14

double povrch_valce(double, double);
double objem_valce(double, double);
double povrch_3(double, double);
double objem_3(double, double);
double povrch_4(double, double);
double objem_4(double, double);
double povrch_6(double, double);
double objem_6(double, double);

double povrch_valce(double vyska, double polomer)
{
    return 2*PI*polomer*vyska+ 2*PI*pow(polomer,2);
}

double objem_valce(double vyska, double polomer)
{
    return PI*pow(polomer,2)*vyska;
}

...
```

Bodovaný úkol

- Napište v jazyku C jednoduchou knihovnu funkcí pro vykreslování obrázků pomocí znaků (tzv. ASCII art).
- Knihovna by měla mít tyto vlastnosti:
 - Obrázky se budou vykreslovat pomocí plátna - dvojrozměrné matice, která bude obsahovat jednotlivé znaky. Vykreslování se tedy neprovádí přímo na výstupu, ale pouze dochází ke změně daného plátna (struktura canvas).
 - Je možné "vykreslovat" i za hranicí kreslící plochy, tyto body se ale nebudou při zobrazení plátna vykreslovat. Jinými slovy, při pokusu o kreslení mimo plátno nedojde k vyjímce při běhu programu.
- Knihovna by měla být samostatným modulem, bude tedy tvořena jedním zdrojovým a jedním hlavičkovým souborem.
- Knihovna bude obsahovat tyto funkce:

```
canvas *canvas_create(int x, int y);
```

- funkce vytvoří plátno

```
void canvas_draw_rect(canvas *c, int x, int y, int width, int height, char ch);
```

- funkce nakreslí na plátno na souřadnice (x,y) obdélník o velikosti width x height znakem ch

```
void canvas_clear(canvas *c);
```

- funkce vyčistí plátno

```
void canvas_print(canvas *c);
```

- funkce vytiskne plátno na obrazovku