Standardní proudy, soubory a datové struktury

IZP-cv05

Ing. Jakub Husa

Vysoké Učení Technické v Brně, Fakulta informačních technologií
Božetěchova 1/2. 612 66 Brno - Královo Pole
ihusa@fit.vut.cz

16. října 2024

Standardní proudy a soubory

Standardní proudy

Základním komunikačním rozhraním programu jsou datové proudy:

- Knihovna stdio.h poskytuje tři standardní proudy (stdin, stdout, stderr).
- stdin – standardní vstup, ze kterého čte funkce scanf (obvykle klávesnice).
- stdout – standardní výstup, do kterého zapisuje funkce printf (obvykle konzole).
- stderr – standardní chybový výstup (obvykle konzole).

Při spuštění z příkazové řádky standardní proudy můžeme přesměrovat na nějaký soubor (nebo program) směrovacími značkami (<, >, >>, 2>, |):

- `./main < input.txt` – vstup z klávesnice nahraď souborem input.txt.
- `./main > output.txt` – výstup do konzole nahraď souborem output.txt.
- `./main >> output.txt` – výstup přidej na konec souboru output.txt.
- `./main 2> error.txt` – chybový výstup přesměruj do souboru error.txt.
- `./main | ./main2` – výstup programu main přesměruj na vstup main2.

Směrovací značka ani název souboru **NEJSOU** součástí argumentů programu:

- Ve VS Code (PowerShell) vstup přesměrujeme příkazem Get-Content

```
Get-Content input.txt | ./main
```

Vyzkoušejte si:

- Vytvořte si soubor input.txt a napište do něj nějaký řetězec.
- Napište program který ze standardního vstupu načte řetězec, obrátí ho, a obrácený řetězec vypíše na standardní výstup.
- Standardní proudy programu přesměrujte do souborů input.txt a output.txt:

  `Get-Content input.txt | ./main > output.txt` (VS code)
- Přihlaste se na server merlin.fit.vutbr.cz, program na něm přeložte s parametry...

  `gcc -std=c11 -Wall -Wextra -Werror main.c -o main`

  ... a vyzkoušejte si že i na něm program funguje:

  `./main < input.txt > output.txt` (merlin)

Například:

Soubor input.txt:

```
1 Ahoj
```

Soubor output.txt:

```
1 johA
```

Otevření a uzavření souboru

Soubor otevřeme funkcí fopen z knihovny stdio.h:

- Vstupem funkce jsou název souboru a režim otevření (`"r"` – read, `"w"` – write).
- Výstupem je proud s datovým typem – `FILE*` – (adresa otevřeného souboru).
- Pokud se soubor nepodařilo otevřít, funkce vrátí chybovou adresu NULL.
- Čtení z nebo zápis do adresy NULL způsobí neplatný přístup do paměti.

Otevřený soubor zavřeme funkcí fclose – vstupem je proud který chceme uzavřít:

- U souborů otevřených pro zápis havárie programu způsobí **ZTRÁTU DAT**!

```
1  FILE* vstup;                        //promenna typu "proud" jmenem "vstup"
2  vstup = fopen("input.txt", "r"); //soubor "input.txt" otevreme pro "cteni"
3
4  if (vstup != NULL)                  //pokud se otevreni podarilo
5  {
6      printf("Otevreni se podarilo\n"); //vypis
7      fclose(vstup);                  //zavirame soubor
8  }
9  else                                //jinak (otevreni selhalo)
10     printf("Otevreni selhalo\n");//vypis
```

Čtení a zápis souboru

Soubory čteme a zapisujeme funkcemi fscanf a fprintf z knihovny stdio.h:

- Funkce používáme stejně jako scanf a printf, ale jako první parametr jim předáváme proud ze kterého mají číst nebo do kterého mají zapisovat.

```
1  FILE* vstup;                              //proud jmenem "vstup"
2  FILE* vystup;                             //proud jmenem "vystup"
3
4  vstup = fopen("input.txt", "r");          //"input.txt" otevirame pro cteni
5  if (vstup != NULL)                        //pokud se otevreni podarilo
6  {
7      vystup = fopen("output.txt", "w"); //"output.txt" otevirame pro zapis
8      if (vystup != NULL)                   //pokud se otevreni podarilo
9      {
10         char x[101];                      //pole znaku
11         fscanf(vstup, "%100s", x);        //z "input.txt" nacitame retezec
12         fprintf(vystup, "%s\n", x);       //a zapisujeme ho do "output.txt"
13
14         fclose(vystup);                   //zavirame soubor "output.txt"
15     }
16     fclose(vstup);                        //zavirame soubor "input.txt"
17 }
```

Návratovou hodnotou funkce scanf je počet úspěšně načtených položek:

- Návratová hodnota umožňuje ověřit úspěšnost načtení vstupu.

```
1  int x, y;                //dve cela cisla "x" a "y"
2  y = scanf("%i", &x);     //do "x" nacteme vstup
3                           //a do "y" priradime navratovou hodnotu
4  if (y > 0)               //pokud se cteni podarilo
5      printf("Cteni se podarilo (x = %i)\n", x);
6  else                     //jinak (pokud cteni selhalo)
7      printf("Cteni selhalo\n");
```

Pokud scanf narazí na konec souboru, vrátí konstantu EOF (end-of-file):

- Na klávesnici EOF zadáme zkratkou ctrl+z (Windows), nebo ctrl+d (Unix).
- Umožňuje nám načítat celý vstupní soubor.

```
8  char y[101];             //vytvarime si pole znaku
9  while (scanf("%100[^\n]\n", y) != EOF) //dokud nenarazime na konec souboru
10 {                        //nacitame retezec (i s mezerami) a konec radku
11     printf("%s\n", y); //vypisujeme nacteny retezec
12 }
```

Na procvičení

Vyzkoušejte si:

- Napište program který jako argumenty programu dostane jména několika souborů, a u každého z nich spočítá kolik obsahuje znaků.
- Pokud se některý ze souborů nepodařilo otevřít, program vypíše hlášení na standardní výstup a bude pokračovat dalším souborem.

Soubor vstup.txt:

```
1 Ahoj
```

Soubor input.txt:

```
1 Ahoj Svete
2 jak se mas
```

Například:

- `./main vstup.txt XYZ input.txt` => Soubor vstup.txt obsahuje 5 znaku
  => Soubor XYZ se nepodarilo otevrit
  => Soubor input.txt obsahuje 22 znaku

Návratová hodnota programu

Návratová hodnota funkce main značí jestli program uspěl nebo selhal:

- Při úspěchu funkce main musí vrátit hodnotu 0 (EXIT_SUCCESS).
- Při selhání funkce main musí vrátit hodnotu jinou něž 0 (EXIT_FAILURE).
- Na příkazové řádce návratovou hodnotu posledního spuštěného programu zjistíme příkazem `echo $?` (Unix) nebo `echo $LastExitCode` (PowerShell).

```
1  int main()                          //zacatek programu
2  {
3      float x;                        //vytvarime si desetinne cislo
4      scanf("%f", &x);                //nacitame do nej vstup
5
6      if (x < 0.0)                    //pokud je cislo zaporne
7          return 1;                   //konec programu - program SELHAL
8
9                                      //jinak spocitame odmocninu
10     printf("sqrt(%f) = %f\n", x, sqrt(x)); //vypis
11     return 0;                      //konec programu - program USPEL
12 }
```

Pokud program selhal, tak měl by také vypsat chybové hlášení:

- Chybová hlášení zapisujeme do proudu stderr (standardní chybový výstup).

```
1  FILE* vstup;                                //"proud" jmenem "vstup"
2
3  vstup = fopen ("input.txt", "r");           //"input.txt" otevirame pro "cteni"
4  if (vstup != NULL)                          //pokud se otevreni podarilo
5  {
6      printf("Otevreni se podarilo\n");       //vypis
7      fclose(vstup);                          //zavirame soubor
8      return 0;                               //program konci bez chyby
9  }
10 else                                        //jinak (otevreni selhalo)
11 {
12     fprintf(stderr, "Chyba: soubor se nepodarilo otevrit\n");
13     return 1;                               //vypis chyboveho hlaseni
14 }                                           //program konci s chybou
```

Datové struktury

Definice struktury a deklarace datového typu

Struktury umožňují spojit spolu položky libovolného datového typu:

- Definice struktury obsahuje klíčové slovo – `struct` –, identifikátor a tělo.
- Tělo struktury obsahuje seznam položek a píšeme za ním středník (;).

```
1  struct Sbod                  //definujeme strukturu jmenem "Sbod"
2  {                            //se dvema polozkami
3      float x;                 //polozka typu "float" jmenem "x"
4      float y;                 //polozka typu "float" jmenem "y"
5  };                           //POZOR -- piseme zde strednik!
```

Klíčovým slovem – `typedef` – pro strukturu deklarujeme nový datový typ:

```
6  typedef struct Sbod bod;     //deklarujeme novy datovy typ pro
7                               //"struct Sbod" jmenem "bod"
```

Datový typ struktury můžeme deklarovat společně s její definicí:

```
8  typedef struct Sbod          //deklarujeme datovy typ pro
9  {                            //"struct Sbod" se dvema polozkami
10     float x;                 //prvni polozka
11     float y;                 //druha polozka
12 } bod;                       //jmeno tohoto typu je "bod"
```

Načítání a vypisování struktur

Struktury načítáme a vypisujeme po jednotlivých položkách:

- K položkám struktury přistupujeme tečkou (.) a identifikátorem.
- Počáteční hodnotu položek můžeme inicializovat složenými závorkami ({}).

```
1  typedef struct Sbod          //definujeme strukturu jmenem "Sbod"
2  {                            //se dvema polozkami
3      float x;                 //polozka typu "float" jmenem "x"
4      float y;                 //polozka typu "float" jmenem "y"
5  } bod;                       //deklarujeme jeji datovy typ "bod"
6
7  int main()                   //zacatek programu
8  {
9      bod A;                   //"bod" jmenem "A" bez pocatecnich hodnot
10     scanf("%f", &A.x);       //nacitame polozku "x" promenne "A"
11     scanf("%f", &A.y);       //nacitame polozku "y" promenne "A"
12     printf("Souradnice bodu A jsou (%f, %f)\n", A.x, A.y);
13
14     bod B = {1.0, 2.0};      //"bod" jmenem "A" s polozkami
15                              //"A.x = 1.0" a "A.y = 2.0"
16     printf("Souradnice bodu B jsou (%f, %f)\n", B.x, B.y);
17     return 0;                //konec programu
18 }
```

Struktury usnadňují organizaci dat a předávání parametrů funkcím:

- Můžeme je používat jako parametry funkcí i návratový typ:
- Pozor – datový typ musíme deklarovat dříve než funkci která ho používá!

```
1  float vzdalenost(bod A, bod B); //deklarace funkce "vzdalenost"
2
3  int main()                      //zacatek programu
4  {
5      bod A = {3.0, 0.0};         //vytvarime Bod A
6      bod B = {0.0, 4.0};         //vytvarime bod B
7      printf("Vzdalenost bodu A a B je %f)\n", vzdalenost(A,B));
8      return 0;                   //konec programu
9  }
10
11 float vzdalenost(bod A, bod B)  //definice funkce "vzdalenost"
12 {
13     float dx = A.x - B.x;       //pocitame rozdil na ose X
14     float dy = A.y - B.y;       //pocitame rozdil na ose Y
15     return sqrt(dx*dx + dy*dy); //pocitame Pythagorovu vetu
16 }
```

Vyzkoušejte si:

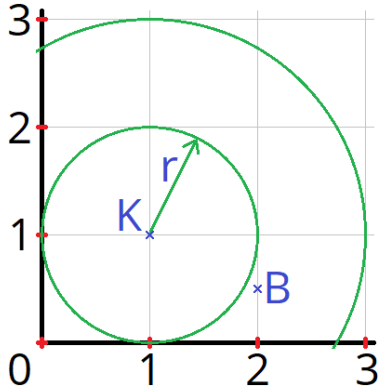
- Vytvořte datový typ kruznice obsahující tři desetinná čísla (x, y, r) představující souřadnice středu kružnice a její poloměr.
- Napište funkci jeUvnitr která ověří jestli se bod nachází uvnitř kružnice.
- Ve funkci main vytvořte kružnici a bod, načtěte do nich nějaké souřadnice, a vypište jestli bod je nebo není uvnitř kružnice.

- Funkce jeUvnitr je deklarována tímto způsobem:

```
1  bool jeUvnitr(kruznice K, bod B);
```

Například:

- ((1.0, 1.0, 2.0), (2.0, 0.5)) => Bod je v kruznici
- ((1.0, 1.0, 1.0), (2.0, 0.5)) => Bod neni v kruznici



Šablona

```
1  //TODO vlozeni knihoven
2  //TODO vytvoreni datoveho typu bod a kruznice
3  //TODO deklarace funkce jeUvnitr
4  int main()                                   //zacatek programu
5  {
6      kruznice K;                              //vytvarime kruznici jmenem "K"
7      bod B;                                   //vytvarime bod jmenem "B"
8
9      scanf("%f", &K.x);                       //souradnice X stredu kruznice K
10     scanf("%f", &K.y);                       //souradnice Y stredu kruznice K
11     scanf("%f", &K.r);                       //polomer R kruznice K
12     scanf("%f", &B.x);                       //souradnice X bodu B
13     scanf("%f", &B.y);                       //souradnice Y bodu B
14
15     if (jeUvnitr(K,B))                       //pokud je bod v kruznici
16         printf("Bod je v kruznici\n");       //vypiseme ze je
17     else                                     //jinak
18          printf("Bod neni v kruznici\n");//vypiseme ze neni
19
20     return 0;                                //konec programu
21 }
22 //TODO definice funkce jeUvnitr
```

Problémy se strukturami

Struktury můžeme přiřazovat pouze pokud mají stejný datový typ:

- Pozor – shodný počet, pořadí a typ položek pro přiřazení nestačí!

```
1  typedef struct Sxxx //deklarujeme datorvy typ pro strukturu
2  {                   //jmenem "Sxxx" s jednou polozkou
3      int cislo;      //prvni polozka
4  } xxx;              //jmeno tohoto typu je "xxx"
5
6  typedef struct Syyy //deklarujeme datorvy typ pro strukturu
7  {                   //jmenem "Syyy" s jednou polozkou
8      int cislo;      //prvni polozka
9  } yyy;              //jmeno tohoto typu je "yyy"
10
11 int main()
12 {
13     xxx A = {10};   //promenna "A" s hodnotou "A.cislo = 10"
14     xxx B = A;      //do promenne "B" prirazujeme promennou "A"
15     yyy C = A;      //CHYBA - do promenne "C" nelze priradit promennou
16     return 0;       //        "A" protoze nejsou stejneho datoveho typu
17 }
```

Zanořené struktury

Položkou struktury může být i další struktura nebo pole:

- K položkám zanořené struktury přistoupíme pomocí několika teček (.).

```
1  typedef struct Susecka    //deklarujeme datovy typ pro strukturu
2  {                         //jmenem "Susecka" se dvema polozkami
3      bod A;                //prvni je typu "bod" a jmenuje se "A"
4      bod B;                //druha je typu "bod" a jmenuje se "B"
5  } usecka;                 //jmeno tohoto typu je "usecka"
6
7  int main()
8  {
9      usecka U;             //vytvarime usecku jmenem "U"
10     scanf("%f", &U.A.x); //nacti desetinne cislo "x" bodu "A" usecky "U"
11     scanf("%f", &U.A.y); //nacti desetinne cislo "y" bodu "A" usecky "U"
12     scanf("%f", &U.B.x); //nacti desetinne cislo "x" bodu "B" usecky "U"
13     scanf("%f", &U.B.y); //nacti desetinne cislo "y" bodu "B" usecky "U"
14
15     printf("Usecka zacina v bode (%.1f, %.1f)\n", U.A.x, U.A.y); //vypis
16     printf("Usecka konci  v bode (%.1f, %.1f)\n", U.B.x, U.B.y); //vypis
17     printf("Delka usecky je %f\n", vzdalenost(U.A, U.B));        //vypis
18     return 0;
19 }
```

Na procvičení

Vyzkoušejte si:

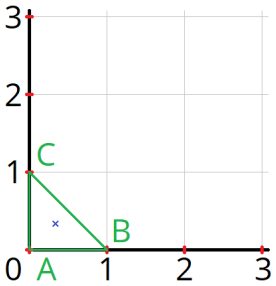
- Deklarujte datový typ trojuhelnik obsahující tři body, představující jeho vrcholy.
- Napište funkce které spočítají obvod, obsah a těžiště daného trojúhelníku.
- Ve funkci main si vytvořte trojúhelník, načtěte souřadnice jeho vrcholů, a vypište výsledek jednotlivých funkcí.

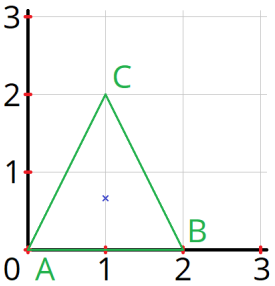
- Funkce jsou deklarovány tímto způsobem:

```
1  float obvod(trojuhelnik T);
2  float obsah(trojuhelnik T);
3  bod teziste(trojuhelnik T);
```

Například:

- ((0.0, 0.0), (1.0, 0.0), (0.0, 1.0)) =>
  => Obvod = 3.414
  => Obsah = 0.500
  => Teziste = (0.333, 0.333)
- ((0.0, 0.0), (2.0, 0.0), (1.0, 2.0)) =>
  => Obvod = 6.472
  => Obsah = 2.000
  => Teziste = (1.000, 0.667)





```c
//TODO vlozeni knihoven a vytvoreni datoveho typu bod
typedef struct Strojuhelnik                 //vytvarime strukturu
{                                           //"Strojuhelnik" s polozkami
    bod A;                                  //  bod "A"
    bod B;                                  //  bod "B"
    bod C;                                  //  bod "C"
} trojuhelnik;                              //struktura je typu "trojuhelnik"
//TODO deklarace funkci
int main()                                  //zacatek programu
{
    trojuhelnik T;                          //vytvarime trojuhelnik jmenem "T"
    scanf("%f %f", &T.A.x, &T.A.y);         //nacitame souradnice vrcholu "A"
    scanf("%f %f", &T.B.x, &T.B.y);         //nacitame souradnice vrcholu "B"
    scanf("%f %f", &T.C.x, &T.C.y);         //nacitame souradnice vrcholu "C"

    printf("Obvod = %.3f\n", obvod(T)); //pocitame a vypisujeme obvod
    printf("Obsah = %.3f\n", obsah(T)); //pocitame a vypisujeme obsah
    bod S = teziste(T);                                 //pocitame teziste
    printf("Teziste = (%.3f, %.3f)\n", S.x, S.y); //vypisujeme teziste
    return 0;                                           //konec programu
}
//TODO definice funkci
```