Programování I

(1. část)

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Ing. Vladimír Beneš, Ph.D. *Petrovický*

vedoucí katedry

K101 – katedra informatiky a kvantitativních metod

E-mail: vbenes@bivs.cz

Telefon: 251 114 534, 731 425 276

Konzultační hodiny: středa 16:00 – 17:30

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Hodinová dotace

- **Prezenční studium**
  - *1 semestr 2/2 Zk 6 kreditů*
- **Kombinované studium**
  - *1 semestr 12/16 Zk 6 kreditů*

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Požadavky ke zkoušce

- **Prezenční studium**
- **Kombinované studium**
  - ***Kompletní vypracování dané úlohy (přihlášení v ISu)***
    - ***Analýza úlohy***
    - ***Algoritmizace***
    - ***Odladěný zdrojový kód programu v jazyce C/C++***

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Studijní literatura

- **Literatura základní**
  - **HEROUT, Pavel. *Učebnice jazyka C*. České Budějovice : Kopp, 1997. ISBN 80-85828-21-9.**
  - **BENEŠ, Vladimír. *Programování I*. Elektronická studijní opora. Praha : BIVŠ, 2011.**
  - **PRATA, Stephen. *Mistrovství v C++*. Brno : Computer Press, 2004. ISBN 80-251-0098-7.**
- **Literatura doporučená**
  - **KADLEC, Václav. Učíme se programovat v jazyce C. Praha : Computer Press, 2002. ISBN 80-7226-715-9.**
  - **MILKOVÁ, E. a kol. *Algoritmy, základní konstrukce v příkladech a jejich vizualizace*. Hradec Králové : Gaudeamus, 2010.**
  - **KNUTH, Donald, E. *Umění programování.1. díl, Základní algoritmy*. Brno : Computer Press, 2008,. ISBN 978-80-251-2025-5.**

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Obsah

- Historie jazyka C/C++
- Syntaxe základních příkazů jazyka C/C++
- Programovací prostředí
- Ladění úlohy

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

PROGRAMOVÁNÍ I

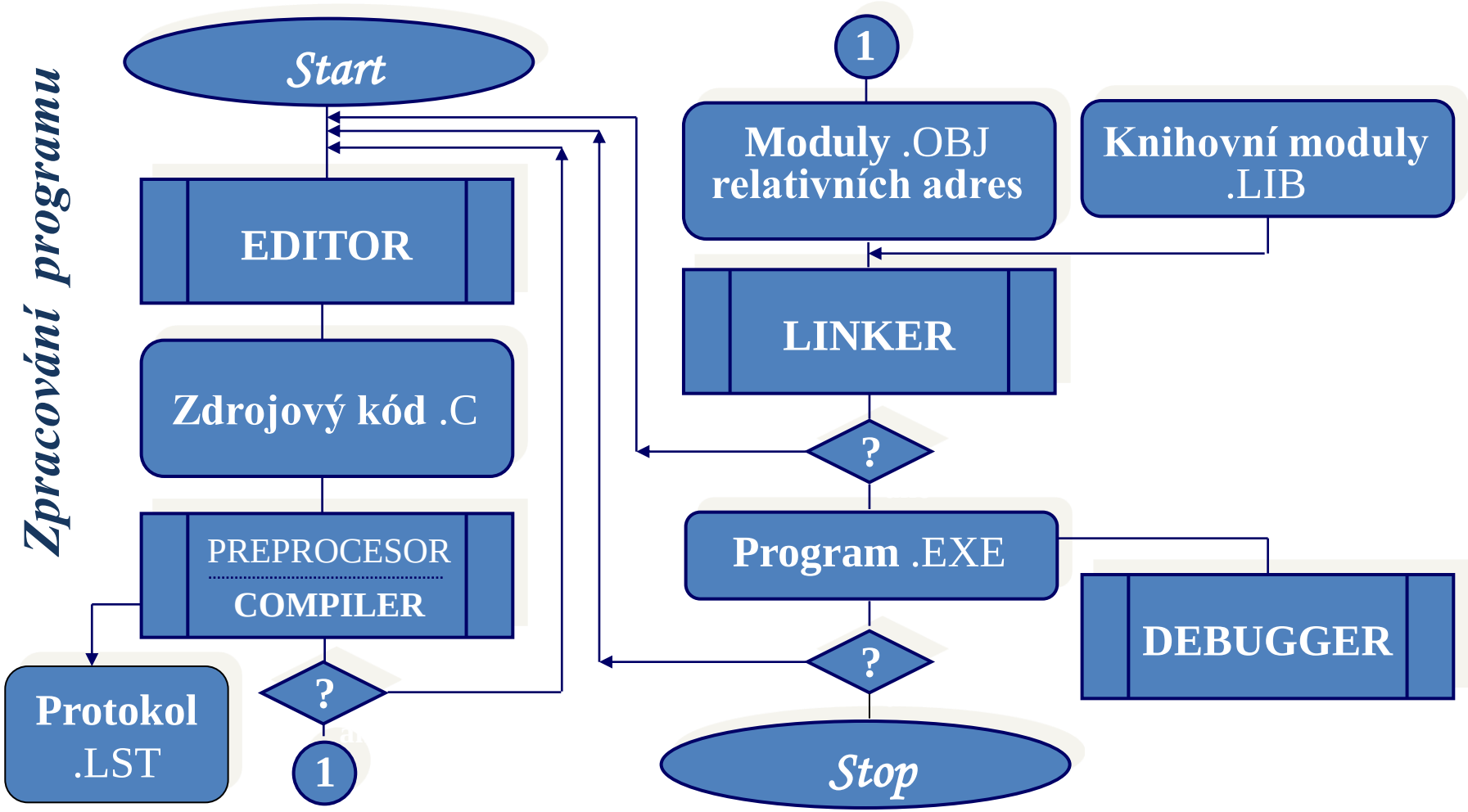
C/C++

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Programování

je proces zahrnující činnosti od:

- **návrhu algoritmu,**
- **psaní,**
- **testování a ladění zdrojového kódu počítačového programu,**
- **včetně následné údržby.**


Zpracování programu
Start
EDITOR
Zdrojový kód .C
PREPROCESOR
COMPILER
Protokol
.LST
?
1
1
Moduly .OBJ
relativních adres
Knihovní moduly
.LIB
LINKER
?
Program .EXE
DEBUGGER
?
Stop

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Jednotlivé programy, kterými je zdrojový text zpracován (1)

Editor — **Příprava a opravy zdrojového textu** (soubor .C).

Preprocesor — **Součást překladače, předzpracovává (upravuje) zdrojový soubor pro kompilátor**.

Compiler — **Překládá zdrojový soubor do relativního kódu počítače** (adresy proměnných a funkcí nejsou ještě známy a jsou zapsány v .OBJ souboru relativně).

Vedlejším produktem je soubor .LIS (protokol).

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Jednotlivé programy, kterými je zdrojový text zpracován (2)

Linker — **Sestavovací program přiřadí relativním adresám adresy absolutní a provede všechny odkazy na dosud neznámé identifikátory (**např. knihovní funkce volané z knihovního souboru .LIB**). Výsledkem práce *linkeru* je spustitelný soubor .EXE.**

Debugger — **Ladící program slouží pro „ladění“, tj. nalézání chyb, které nastávají při běhu programu (**„odvšivovač“**).**

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Základní pojmy v jazyce C/C++

Zdrojové a hlavičkové soubory

- zdrojový soubor .C nutno „doplnit“ vložením souborů
- hlavičkové soubory (*header*) .h
  - #include <stdio.h>
  - #include <conio.h>
  - #include <math.h>
  - …

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Základní pojmy v jazyce C/C++

Štábní kultura

- snadná orientace ve zdrojovém textu programu
- bílé znaky (*white spaces*) = oddělovací znaky
  - mezera
  - tabelátor
  - nový řádek
  - …

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Základní pojmy v jazyce C/C++

Identifikátory

- jazyk C je *case sensitive* jazyk
  - rozlišují se malá a velká písmena
    - prom, Prom, PROM
  - klíčová slova (if, for, …) musí být malými písmeny
  - je možné používat *podtržítko* _
    (_prom, prom_x, prom_)
  - délka identifikátoru není omezena
    (ANSI C rozeznává 32 znaků)

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Základní pojmy v jazyce C/C++

„Štábní“ kultura

- běžně používané významové identifikátory
  - i, j, k - indexy, parametry cyklů
  - c, ch - znaky
  - m, n - čítače
  - f, r - reálná čísla
  - p_ - začátek identifikátoru pointeru (ukazatel)
  - s - řetězec

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Základní pojmy v jazyce C/C++

„Štábní“ kultura

- běžně používané významové identifikátory
  - i, j, k - indexy, parametry cyklů
  - c, ch - znaky
  - m, n - čítače
  - f, r - reálná čísla
  - p_ - začátek identifikátoru pointeru (ukazatel)
  - s - řetězec

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Základní pojmy v jazyce C/C++

Komentáře

- **přehlednost programu (*ale i ladící prvek*)**
  - **nedoplňovat komentáře, „až zbude čas“**
  - **// *jednořádkový* komentář**
  - **/* toto je komentář *blokový* */**
  - **/* toto je komentář /* toto je vložený komentář*/ */**
  - **/***

    *** výrazný víceřádkový komentář**

    ***/**

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Jednoduché datové typy a přiřazení

Pascal	C
INTEGER	int long int short int
CHAR	char
REAL	float double long double

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Jednoduché datové typy a přiřazení

Definice proměnných

- příkaz, který udělí proměnné určitého typu jméno a paměť
  - globální (vně funkce)
  - lokální (uvnitř funkce)

Deklarace proměnných

- příkaz, který pouze udává typ proměnné a její jméno

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Jednoduché datové typy a přiřazení

Definice proměnných

Pascal	C
VAR i : INTEGER;	int i;
c : CHAR;	char c, ch;
f, g : REAL;	float f, g;

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Jednoduché datové typy a přiřazení

Definice proměnných

```
int  i;                          /* globální proměnná */

main()
   {
      int  j;                    /* lokální proměnná */
   }
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Jednoduché datové typy a přiřazení

Přiřazení

česky	anglicky	symbolicky	prakticky
výraz	*expression*	výraz	i * 2 + 3
přiřazení	*assigment*	l-hodnota = výraz	j = i * 2 + 3
příkaz	*statement*	l-hodnota = výraz;	j = i * 2 + 3;

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Jednoduché datové typy a přiřazení

Několikanásobné přiřazení

```
k = j = i = 2;
```

vyhodnocuje se zprava doleva, tedy:

```
k = (j = (i = 2));
```

Hlavní program (= hlavní funkce)

```
main()                          /* bez středníku !!! */
 {
      int  i, j;

      i = 5;
      j = -1;
      j = j +2 * i;
 }
```

Hlavní program (= hlavní funkce)

TÉŽ MOŽNO inicializovat proměnné přímo v definici

```
main()                          /* bez středníku !!! */
 {
      int       i = 5,
                j = -1;

      j = j +2 * i;
 }
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Konstanty

Celočíselné

- *dekadické* (posloupnost číslic, z nichž první nesmí být 0)
- *oktalové* (číslice 0 následovaná posl. oktal. číslic (0 – 7)
- *hexadecimální* (číslice 0 následovaná znakem x (nebo X) a posloupností hexadecimálních číslic (0 – 9, a – f, A – F)

• dekadické	15, 0, 1
• oktalové	065, 015, 0, 01
• hexadecimální	0x12, 0X3A, 0x0, 0x1, 0Xcd

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Konstanty

Reálné

- mohou začínat a končit tečkou
- implicitně jsou typu *double*

  15. 156.88 .84 3.14 5e6 7E23
- konstanta typu *float* se definuje pomocí přípony f (nebo F)

  3.14f (nebo 3.14F)
- konstanta typu *long double* se definuje pomocí přípony l (nebo L)

  12e3l (nebo 12E3L)

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Konstanty

Znakové

- hodnota znakové konstanty (ordinální číslo) je odvozena z odpovídající kódové tabulky (*ASCII*)
- velikost znakové konstanty je typu *int* (ne *char*)
- jsou uzavřeny mezi apostrofy
  ‘a‘ ‘*‘ ‘4‘
- zápis neviditelné konstanty
  ‘\012‘ ‘\007‘ (tzv. *escape* sekvence)

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Konstanty

Escape sekvence

sekvence	hodnota	význam
\n	0x0A	*new line* (nová řádka)
\r	0x0D	*carriage return* (návrat na zač. ř.)
\f	0x0C	*formfeed* (nová řádka)
\t	0x09	tabulátor
\b	0x08	*backspace* (posun doleva)
\a	0x07	*BELL* (písknutí)

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Konstanty

Escape sekvence

sekvence	hodnota	význam
\\	0x5C	*backslash* (zpětné lomítko)
\‘	0x2C	*single quote* (apostrof)
\0	0x00	*nul character* – NUL (nulový zn.)

Poznámka: NUL není NULL (nulový pointer)

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Konstanty

Řetězcové konstanty (*literály*)

- **v řetězcových konstantách se používá pro zobrazení znaku uvozovky (*double quote*) *escape* sekvence \“**

  ale

- **uvozovky jako znaková konstanta: ‘ “ ‘**
- **“Toto je řetězcová konstanta“**

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Konstanty

Řetězcové konstanty (*literály*)

- ekvivalentní zápis dlouhé řetězcové konstanty
  - “Velmi dlouhý řetězec znaků“
  - “Velmi dlouhý“ “ řetězec znaků“
  - “Velmi“ “ dlouhý“ “ řetězec znaků“

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Aritmetické výrazy

Výraz ukončený středníkem se stává příkazem!!!

- **i = 2** **výraz s přiřazením**
- **i = 2;** **příkaz**

- **pouhý středník = prázdný příkaz (*null statement*)**
  - **pozor na použití v příkazech cyklu *for* nebo *while***

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Aritmetické výrazy

Unární operátory

- **unární plus** +
- **unární mínus** -

- **oba operátory se používají v běžném významu**

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Aritmetické výrazy

Binární operátory

- sčítání +
- odčítání -
- násobení *
- reálné dělení /
- celočíselné dělení / (záleží na typu operandů)
- dělení *modulo* %

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Aritmetické výrazy

Speciální unární operátory

- *inkrement* ++
- *dekrement* --
- *Pozor*: výraz musí být l-hodnota, tedy *proměnná*
  - nelze tedy
    - 45++
    - --(i + j)

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Aritmetické výrazy

Speciální unární operátory

- oba operátory se dají použít jako předpona (*prefix*), ale také jako přípona (*surfix*)
  - ++výraz (*inkrementování před použitím*)
    - výraz je nejprve zvětšen o 1 a pak je nová hodnota vrácena jako hodnota výrazu
  - výraz++ (*inkrementování po použití*)
    - je vrácena původní hodnota výrazu a pak je výraz zvětšen o 1

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Aritmetické výrazy

Speciální unární operátory

- např.:

```
int  i = 5,  j = 1,  k;

i++;                    // i bude 6

j = ++i;                // i bude 7, j bude 7

j = i++;                // j bude 7, i bude 8

k = - -j + 2;           // k bude 8, j bude 6 (i bude 8)
```

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Terminálový vstup a výstup

Vstup a výstup znaku

- nutný hlavičkový soubor stdio.h
  - #include <stdio.h>
  - funkce putchar() *výstup jednoho znaku*
  - funkce getchar() *vstup jednoho znaku*

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Terminálový vstup a výstup

PŘÍKLAD

```
#include <stdio.h>   // vstup a výstup znaku

void  main()
  {
     int c;

     c = getchar();
     putchar(c);
     putchar('\n');
  }
```

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Formátovaný vstup a výstup

- funkce *scanf*() *pro formátovaný vstup*
- funkce *printf*() *pro formátovaný výstup*
  - proměnný počet parametrů

    (1x *řídící řetězec formátu*
    a
    seznam k proměnných) $k = 0, 1, 2, 3, \ldots$

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Formátovaný vstup a výstup

- řídící řetězec formátu obsahuje
  - formátové specifikace
    - začínají znakem „%“
    - určují formát vstupu, resp. výstupu
  - znakové posloupnosti
    - nezačínají znakem „%“
    - vypíší se tak, jak jsou zapsány
    - je možné použít českou diakritiku
    - používají se pouze pro funkci *printf*()

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Formátovaný vstup a výstup

formátové specifikace uváděné se znakem „%“ (1)

c	**znak**
d	**dekadické číslo *signed int***
ld	**dekadické číslo *signed long***
u	**dekadické číslo *unsigned int***
lu	**dekadické číslo *unsigned long***
f	***float***
lf	***double***
Lf	***long double***

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Formátovaný vstup a výstup

formátové specifikace uváděné se znakem „%“ (2)

x	**hexadecimální číslo malými písmeny (1a2c)**
X	**hexadecimální číslo velkými písmeny (1A2C)**
o	**oktalové číslo**
s	**řetězec (*string*)**

---

i	***int***

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Formátovaný vstup a výstup

PŘÍKLAD

```
…
float a, b, c;

printf(“\n Řešíme kvadratickou rovnici“);
printf(“\n Zadej parametry a, b, c:  “);
scanf (“%f %f %f“, &a, &b, &c);
…
```

!!! operátor &, resp. | jsou tzv. *bitové operátory*

Programování I

(2. část)

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Boolovské výrazy

- v jazyce C/C++ není implicitně typ Boolean
- místo tohoto typu se používá typ *int*
  - nulová hodnota znamená hodnotu *FALSE*
  - nenulová hodnota (nejčastěji = 1) znamená *TRUE*

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Logické operátory

Pascal	C	
=	==	*rovnost*
<>	!=	*nerovnost*
AND	&&	*logický součin*
OR	||	*logický součet*
NOT	!	*negace*

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Relační operátory

Pascal	C	
<	<	*menší*
<=	<=	*menší nebo rovno*
>	>	*větší*
>=	>=	*větší nebo rovno*

Řídící struktury

Priority vyhodnocování logických výrazů 1

operátor	*směr vyhodnocení*
! -- ++ - + (typ)	zprava doleva
* / %	zleva doprava
+ -	zleva doprava
< <= >= >	zleva doprava
== !=	zleva doprava

Řídící struktury

Priority vyhodnocování logických výrazů (2)

operátor	*směr vyhodnocení*
&&	zleva doprava
||	zleva doprava
? :	zprava doleva
= += -= *= atd.	zprava doleva
,	zleva doprava

!!! Tabulka není úplná; obsahuje nejčastější operátory !!!

Řídící struktury

Podmíněný výraz

syntaxe: *výraz_podm* **?** *výraz_1* **:** *výraz_2*

význam: **if** *výraz_podm* **then** *výraz_1* **else** *výraz_2*

Příklad:

```
int  i, k, j = 2;

i = (j==2)  ?  1  :  2;    /* i  bude 1 */
k = (i > j)  ?  i   :  j;    /* k bude max. z i a j, tedy 2 */
```

Řídící struktury

Operátor čárky

syntaxe: *výraz_1 , výraz_2*

význam: vyhodnotí se *výraz_1*, je zapomenut a vyhodnotí se *výraz_2* a ten je výsledkem; není to *l_hodnota*

Příklad:

```
int  i = 2, j = 4;          /* toto není operátor čárky */
j = (i++ ,  i – j);         /* i bude 3,  j bude -1 */
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Upozornění

!!! Pouze 4 operátory v C/C++ zaručují vyhodnocení levého operandu před vyhodnocením pravého operandu !

Jsou to:

logický součin	**&&**
logický součet	**||**
ternární operátor	**? :**
operátor čárky	**,**

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Příkaz *if*

syntaxe: if(*výraz_podmínka*) *příkaz*;

význam: Platí-li *výraz_podmínka*, tj. *výraz_podmínka* má hodnotu ≠ 0, provede se *příkaz*, jinak se jde dál

Příklad:

```
int  c;
if((c = getchar()) >= 'A'  &&  c <= 'Z')  printf("\n %i", c);
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Příkaz *if-else*

syntaxe: **if(*výraz_podmínka*) *příkaz_1*;**
else *příkaz_2*;

význam: **Platí-li *výraz_podmínka*, tj. *výraz_podmínka* má hodnotu ≠ 0, provede se *příkaz_1*, jinak příkaz_2**

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Příkaz *if-else*

syntaxe: **if(*výraz_podmínka*) *příkaz_1*;**
else *příkaz_2*;

Příklad 1:

```
if(i > 3)
        j = 5;
  else
        j = 1;
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Příkaz *if-else*

Příklad 2:

```
if(i > 3)
    {
     j = 5;
     k = 4;
    }
  else
    {
     j = 5;
     k = 4;
    }
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Iterační příkazy - cykly

Příkazy *break* a *continue*

Oba příkazy lze použít ve všech třech typech cyklů.

break	**ukončuje nejvnitřnější neuzavřenou smyčku; opouští okamžitě cyklus**
continue	**skáče na konec nejvnitřnější neuzavřené smyčky a tím vynutí další iteraci smyčky; cyklus neopouští**

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Iterační příkazy - cykly

Příkaz *while*

syntaxe: **while (*výraz_podmínka*)**
***příkaz*;**

Tento iterační příkaz testuje podmínku cyklu před průchodem cyklem.

Cyklus tedy nemusí proběhnout ani jednou.

Iterační příkazy - cykly

Příkaz *while*

Příklad 1:

```
while (x < 10)
    x++;
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Iterační příkazy - cykly

Příkaz *while*

Příklad 2:

```
while (x < 10)
  {
     x++;
     y = 2*x + 15;
     z = x – y;
  }
```

Iterační příkazy - cykly

Příkaz *while*

Příklad 3:

```
int c;
while (1)                    /* nekonečná smyčka */
    {
      if((c = getchar()) < ' ')
        continue;            // zahoď „bílý“ znak
      if(c == 'z')
        break;               // celkové ukončení
      putchar(c);            // tisk znaku
    }
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Iterační příkazy - cykly

Příkaz *do-while*

syntaxe: **do** ***příkaz*;**
while (*výraz_podmínka*)

Tento iterační příkaz testuje podmínku cyklu až po průchodu cyklem.

Cyklus tedy musí proběhnout nejméně jednou.

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Iterační příkazy - cykly

Příkaz *do-while*

Příklad 1:

```
do
    i--;
while (i > 0);
```

Iterační příkazy - cykly

Příkaz *do-while*

Příklad 2:

```
int  c;
...
do
    {
      if((c = getchar()) >= ' ')
          putchar(c);
    }
  while(c != 'z');
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Iterační příkazy - cykly

Příkaz *for*

syntaxe: `for(výraz_začátek; výraz_konec; výraz_krok)`
`příkaz;`

Tento příkaz cyklu použijeme, známe-li předem počet průchodů cyklem.

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Iterační příkazy - cykly

Příkaz *for*

Příklad 1:

```
for(i = 0; i < 10; i++)
    printf("\n %i", i);
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Iterační příkazy - cykly

Příkaz *for*

Příklad 2:

```
int  i, soucin;
…
for(i = 3, soucin = 1; i <= 9; i += 2)
    soucin *= i;
```

Iterační příkazy - cykly

Příkaz *for*

Příklad 3:

```
for( ; ; )          /* nekonečný cyklus*/
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Příkaz *switch*

syntaxe: `switch(`*výraz*`)`

```
{
    case hodnota_1 : příkaz_1;     break;
    case hodnota_2 : příkaz_2;     break;
    …
    case hodnota_n : příkaz_n;     break;
    default        : příkaz_def;   break;
}
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Příkaz *switch*

Příklad:

```
switch(getchar())
   {
      case ‘a‘      : putchar(‘1‘); break;
      case ‘b‘      : putchar(‘2‘); break;
      case ‘c‘      : putchar(‘3‘); break;
      case ‘d‘      : putchar(‘4‘); break;
      default       : putchar(‘0‘); break;
   }
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Příkaz *goto* **NEPODMÍNĚNÝ SKOK**

syntaxe: goto *návěští*;
…
návěští: příkaz;

V programu je předáno řízení na příkaz s návěštím.

Návěští je *identifikátor*.

Příkaz *goto* se v dobře napsaných prog. používá řídce; ve strukturovaném jazyku se mu lze vyhnout.

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Příkaz *goto*

***Příklad*:**

```
                for(i = 1; i < 10; i++)
                    for(j = 1; j < 10; j++)
                        {
                            if(x == 0)
                                goto error;
                        }
                goto další_výpočet;
error:          printf( … );
další_výpočet:  …
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Příkaz *return*

syntaxe: **return (*výraz*);**

Příkaz *return* ukončí provádění funkce, která tento příkaz obsahuje.

Ve funkci *main* ukončí příkaz *return* celý program.

Pomocí příkazu *return* se vrací hodnota, jejíž typ záleží na typu funkce (na typu *návratové hodnoty*).

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Řídící struktury

Příkaz *return*

Příklad:

```
…
return(0);     /* neúspěch */
…
return(1);     /* úspěch */
…
```

Programování I

(3. část)

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Pole

POLE

- homogenní struktura
- deklarace
  - statická (*velikost známá už při překladu !!!*)
  - dynamická (*požadavek na paměťový prostor kladen během běhu programu*)
  - prvky pole určeny indexováním
  - v C/C++ indexováno vždy od nuly

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Pole

POLE

- **jednorozměrná** (***vektor***)
- **dvourozměrná** (***matice***)
- **třírozměrná** (***kubická matice***)
- **…**
- **vícerozměrná**

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Pole

STATICKÁ ALOKACE POLE V PAMĚTI

datový_typ ident_pole [počet];

***Příklad*:**

```
float x[10], y[15];  // jednorozměrné pole;
                        pole x má 10 prvků: x(0), … , x(9)

int   a[10][20];     // celoč. matice; má 10 řádků a 20 sloupců
                        prvky      a(0,0), … , a(0,19)
                                   …
                                   a(9,0), … , a(9,19)
```

Pole – statická alokace paměti

PRÁCE S PRVKEM JEDNOROZMĚRNÉHO POLE

```
int k;
float x[10], soucin;
…                          // součin prvků vektoru x  ( ∏_{k=1}^{10} x_k )
soucin = 1;
for(k=0; k<10; k++)
     soucin = soucin * x[k];
```

Pole – statická alokace paměti

PRÁCE S PRVKEM DVOUROZMĚRNÉHO POLE

```
int i;
float a[10][10], stopa;
…                                   // stopa matice A = ( Σ_{i=1}^{10} a_ii )
stopa = 0;
for(i=0; i<10; i++)
     stopa = stopa + a[i][i];
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Pole – statická alokace paměti

POJMENOVANÁ KONSTANTA

#define IDENT_KONSTANTY hodnota

***Příklad*:**

#define PI 3.14159

***Poznámka*: IDENT_KONSTANTY – velká písmena**

Pole – statická alokace paměti

PŘÍKLAD

```
#define  X_MAX  10

int k, n;
float x[X_MAX], soucin;
…                          // součin prvků vektoru x  ( ∏_{k=1}^{10} x_k )
soucin = 1;
for(k=0; k<n; k++)                                  //  n <= X_MAX !!!
     soucin = soucin * x[k];
```

Pole – statická alokace paměti

PŘÍKLAD

```
#define  N_MAX  10

int i;
float a[N_MAX][N_MAX], stopa;
…                                   //  stopa matice A  ( Σ_{i=1}^{10} a_ii  )
stopa = 0;
for(i=0; i< n; i++)                 //  n <=  N_MAX  !!!
     stopa = stopa + a[i][i];
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Pole – dynamická alokace paměti

DYNAMICKÁ ALOKACE PAMĚTI

malloc(*velikost*) // knihovna <stdlib.h>, resp. <alloc.h>

sizeof (xxx) // operátor, který zjistí velikost zkoumaného datového objektu v bytech

xxx // zkoumaný datový objekt

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Pole – dynamická alokace paměti

DYNAMICKÁ ALOKACE PAMĚTI

sizeof(pole)	**vrací velikost proměnné pole**
sizeof(*pole)	**vrací velikost pointeru na proměnnou pole**
sizeof (double)	**vrací velikost typu *double***
sizeof(double*)	**vrací velikost pointeru na typ *double***

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Pole – dynamická alokace paměti

DYNAMICKÁ ALOKACE jednorozměrného pole

***Příklad*:**

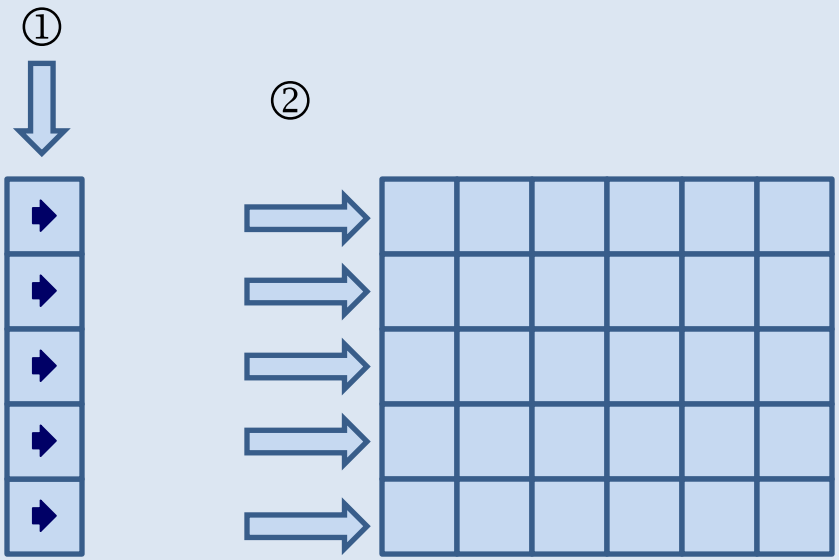
```
#include <stdlib.h> //standardní knihovna, resp. <alloc.h>

double  *v;        //  deklarace jednorozměrného pole,
                       hvězdička (*) označuje pointer (ukazatel)

v = (double*) malloc(n * sizeof(double));
```

Pole – dynamická alokace paměti

DYNAMICKÁ ALOKACE dvourozměrného pole

***Schéma*:**

double **a;



BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Pole – dynamická alokace paměti

DYNAMICKÁ ALOKACE dvourozměrného pole

***Příklad*:**

```
int       i, n;
double  **a;    //  deklarace dvourozměrného pole,
                    hvězdička (*) označuje pointer (ukazatel)
                    na jednorozměrné pole (*) pointerů (ukazatelů)
...
a = (double**) malloc(m * sizeof(double*));
for(i = 0; i<m; i++)
        a[i] = (double*) malloc(n * sizeof(double));
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Pole – dynamická alokace paměti

UVOLNĚNÍ PAMĚTI

free(); // parametrem funkce je pointer na typ *void*, který ukazuje na začátek dříve přiděleného bloku

***Příklad*:**

```
float **a;
…
free(a);
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Pole – dynamická alokace paměti

TESTOVÁNÍ ÚSPĚŠNOSTI ALOKACE PAMĚTI

```
char      *p_c;
…
*p_c = ´a´;
```

Takový příkaz není zcela korektní .
p_c ukazuje někam do paměti a my ji nemáme přidělenou!

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Pole – dynamická alokace paměti

TESTOVÁNÍ ÚSPĚŠNOSTI ALOKACE PAMĚTI

```
char      *p_c;                    // správné řešení problému
…
if ((p_c = malloc(1)) == NULL)
  {
     printf("\n Není dostatek volné paměti!");
     return;
   }
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Uživatelská funkce

STRUKTURA JAKO U FUNKCE main

```
#include …
```

y = fce (x_1, x_2, …, x_n)

```
void main()
 {
   // dekl. lok. prom.
   // výpočet
 }
```

```
[typ_návrat_hod] id_fce(sez. form. par.)
 {
   // dekl. lok. prom.
   // výpočet
   [return ();]
 }
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Uživatelská funkce

SEZNAM FORMÁLNÍCH PARAMETRŮ

… (dat_typ $prom_1$, dat_typ $prom_2$, … , dat_typ $prom_n$)

Seznam *formálních* parametrů může být ***prázdný*** !

Uživatelská funkce

VOLÁNÍ UŽIVATELSKÉ FUNKCE

```
dat_typ_navrat_hod   prom;

…
prom = id_fce(seznam skutečných parametrů);
…
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Uživatelská funkce

SEZNAM SKUTEČNÝCH PARAMETRŮ

… $(prom_1 , prom_2 , … , prom_n)$

Seznam *skutečných parametrů* může být *prázdný* !

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Uživatelská funkce

VZTAH FORMÁLNÍCH A SKUTEČNÝCH PARAMETRŮ

- odpovídající
  - *počet* parametrů
  - *pořadí* parametrů
- jde o *volání hodnotou* (není-li uveden pointer)

Uživatelská funkce

PŘÍKLAD pokr. 0

```
#include  < … >
…
float  suma(int a, float b, float c)  // deklarace uzivatelske fce
  {
    float  soucet;
    …
    soucet = a + b + c;
    return  (soucet);
  }
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Uživatelská funkce

PŘÍKLAD pokr. 1

```
void main()
  {
    int    aa;
    float bb, cc, vysledek;
    …
    vysledek =  suma(aa, bb, cc);   // volání už. fce – skut. par.
    …
  }
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Uživatelská funkce

PARAMETREM UŽIVATELSKÉ FCE - POLE

- pole se chová jako „pointer“ (ukazatel)
- jde o *volání* parametru *odkazem*

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Uživatelská funkce

PARAMETREM UŽIVATELSKÉ FCE - POLE

```
… ( … , dat_typ *v, dat_typ**a, …) // formální parametry
…
…
float *u, **b;
…
… ( … u, b, …)                    // skutečné parametry
…
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Uživatelská funkce

PARAMETREM UŽIVATELSKÉ FCE - FUNKCE

```
dat_typ id_f1(form. parametry uživatelské funkce)
…
dat_typ id_f2(form. parametry uživatelské funkce)
  …
dat_typ id_f( … ,  (dat_typ ) (*fce) (dat_typ) … ) // form. param.
  …
…
prom = f(…, f1, …);          // volaní už. funkce – skutečné parametry
…
prom = f(…, f2, …);          // volaní už. funkce – skutečné parametry
…
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Uživatelská funkce

PARAMETREM UŽ. FCE - VÝSTUPNÍ PARAMETR

- jde o volání ODKAZEM
- použije se symbol * (*pointer*) v
  - v záhlaví deklarace uživatelské funkce
  - v těle deklarace uživatelské funkce

Uživatelská funkce

PŘÍKLAD pokr. 0

```
void rosada(int *p_a, int *p_b)
 {
    int pom;
    pom   =  *p_a;
    *p_a  =  *p_b;
    *p_b  =  pom;
 }
```

Uživatelská funkce

PŘÍKLAD pokr. 1

Volání funkce *rošáda* se skutečnými parametry:

```
int  k = 10,  l = 20;

…
rosada(&k,  &l);     // pozor na uvedení „adresy“
…
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Práce se souborem

1. **DEKLARACE PROMĚNNÉ TYPU SOUBOR**

   FILE id_file;

Příklad

FILE *fr, *fw;

***Poznámka*:**

- **deklarujeme dvě proměnné typu soubor**
- **např.**
  - **fr - bude vstupní soubor (read)**
  - **fw - bude výstupní soubor (write)**

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Práce se souborem

2. OTEVŘENÍ SOUBORU pokr. 1

id_file = fopen(“*cesta*“, “*atribut*“);

tj. přiřazení symbolickému souboru* id_file *soubor fyzický

***cesta* = řetězec (fyzické umístění a jméno souboru)**

***atribut* = znak**

- **r = textový soubor pro čtení**
- **w = textový soubor pro zápis nebo přepsání**
- **a = textový soubor pro připojení na konec**

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Práce se souborem

2. OTEVŘENÍ SOUBORU pokr. 2

atribut = znak

- rb = binární soubor pro čtení
- wb = binární soubor pro čtení nebo přepsání
- ab = binární soubor pro připojení na konec

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Práce se souborem

2. OTEVŘENÍ SOUBORU pokr. 3

atribut = znak

- r+ = textový soubor pro čtení a zápis
- w+ = textový soubor pro čtení, zápis nebo přepsání
- a+ = textový soubor pro čtení a zápis na konec

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Práce se souborem

2. OTEVŘENÍ SOUBORU pokr. 4

atribut = znak

- rb+ = binární soubor pro čtení a zápis
- wb+ = binární soubor pro čtení zápis nebo přepsání
- ab+ = binární soubor pro čtení a zápis na konec

- analogicky též pro textový soubor
  možno „“rt“, “wt“, “at“

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Práce se souborem

3. OTESTOVÁNÍ EXISTENCE SOUBORU

např.:

```
if ((fw = fopen(file_output, “w“) == NULL)
    {
      …
    }
```

BANKOVNÍ INSTITUT
VYSOKÁ ŠKOLA

Práce se souborem

4. PRÁCE NA SOUBORU

fscanf (id_file, "formátová specifikace", seznam_proměn.)
pro čtení ze souboru

fprintf (id_file, "formátová specifikace", seznam_proměn.)
pro zápis do souboru

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Práce se souborem

5. UZAVŘENÍ SOUBORU

```
fclose(id_file);
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Práce se souborem

PŘÍKLAD 1

```
#include <stdio.h>
void main()
 {
    FILE *fw;
    int  i;

    fw = fopen("data_o.txt", "w");
      for(i=1; i<=10; i++)
          fprintf(fw, "%d \n", i);
    fclose(fw);
 }
```

BANKOVNÍ INSTITUT VYSOKÁ ŠKOLA

Práce se souborem

PŘÍKLAD 2

```
#include <stdio.h>
void main()
 {
    FILE *fr;
    double  x, y, z;

    fr = fopen("data_i.txt", "r");
      fscanf(fr, "%lf %lf %lf", &x, &y, &z);
      printf("%f \n", x + y + z);
    fclose(fr);
 }
```

Děkuji za pozornost