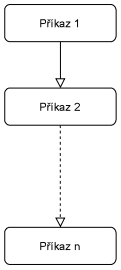
Řízení běhu programu

Programy, které jsme doposud vytvářeli, byly vykonávány *sekvenčně*, tedy postupně se prováděly příkazy ve funkci `main()`. Průběh programu je znázorněn na obrázku 1.
Pořadí, ve kterém se příkazy v programu vykonávají, se nazývá *tok programu* (program flow). Kromě sekvenčního vykonávání (*sekvence*), je možné tok programu měnit pomocí větvení (*selekce*) a cyklů (*iterace*). Právě těmto konstrukcím se tuto kapitolu budeme věnovat.

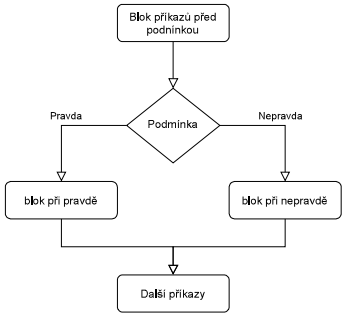
Obrázek 1: Grafické znázornění průběhu programu.



Větvení

Větvení programu umožňuje rozhodnout na základě *podmínky*, jaká část programu se bude vykonávat. Větvení se skládá z logického výrazu reprezentujícího podmínku a bloků příkazů, které se vykonávají v závislosti na tom, zda je nebo není podmínka splněna. Průběh programu je znázorněn na obrázku 2.

Obrázek 2: Grafické znázornění průběhu programu obsahujícího větvení.



Větvení pomocí `if` a `else`

Pro zápis větvení programu se v jazyce C používají příkazy `if` a `else`. Zápis vypadá následovně.

```
if (podminka)
    blok_pri_pravde
else
    blok_pri_nepravde
```

Přičemž `else` část je možné vynechat.
`podminka` je logický výraz. Pokud je podmínka splněna (je pravdivá), provede se `blok_pri_pravde`. Pokud není, vykoná se `blok_pri_nepravde`.
Jak už víme, jazyk C neobsahuje logický datový typ. Hodnota `0` představuje *nepravdu* (ať je jakéhokoliv typu) a ostatní hodnoty se vyhodnocují jako *pravda*.
Většina podmínek se skládá z logických binárních operátorů:

- $<, <=$ menší, menší rovno

- $>, >=$ větší, větší rovno
- $==$ rovnost (pozor nezaměňovat s operátorem přiřazení $=$)
- $!=$ nerovnost

Podmínky je možné spojovat do složitějších podmínek pomocí logických operací

- `||` nebo
- `&&` a zároveň
- `!` negace

Logické operátory `||` a `&&` se vyhodnocují *líně*. Pokud je levý argument `||` pravdivý, druhý argument se už nevyhodnocuje (není potřeba ho vyhodnocovat, výsledek je vždy pravda). Obdobně u `&&` pokud je první argument nepravda, není potřeba vyhodnocovat druhý (výsledek je vždy nepravda). Příklad vhodného použití je následující.

```
if ((y != 0) && ((x / y) < z))
```

Výraz je zcela správný a nikdy nedojde k dělení nulou. Pokud by bylo y rovno 0, pak první část výrazu `y != 0` ukončí jeho vyhodnocování.
V následujícím kódu je příklad programu, který pro dvě zadaná čísla vypíše větší z nich.

```
if (a >= b){
    printf("%d", a);
}
else{
    printf("%d", b);
}
```

Podmínky je vhodnější kvůli přehlednosti psát v kladném znění tvrzení a ne v negaci, obzvláště ve složených výrazech. Například následující výraz není úplně vhodný.

```
/* nevhodne */
if(!(c == '\0' || c == ' ' || c == '1')
```

Než si ukážeme řešení, jak by stejná podmínka měla vypadat v kladném znění, zkuste se nad tím zamyslet sami. Zkuste výraz upravit tak, aby neobsahoval negaci na začátku.
Lepší zápis stejné podmínky je následující.

Úkol 14
Co vypíše následující část kódu?

```
int a = 0;

if (a = 0){
    printf("a je rovno 0"
        );
}
else{
    printf("a neni rovno
        0");
}
```

Vyzkoušejte, zda máte pravdu.

```
/* vhodnejsi */
if (c != '\0' && c != ' ' && c != '1')
```

Vnoření větvení

V kódu je možné do sebe jednotlivá větvení vnořovat. Bloky `blok_pri_pravde` i `blok_pri_nepravde` mohou obsahovat další `if` konstrukce. Pokud blok obsahuje jen jediný příkaz, je možné vynechat ohraničující závorky `{ }`, ale nedoporučuje se to, obzvláště u vnořených větvení, abychom dali najevo, které `if` a `else` patří k sobě. Bez použití závorek `else` vždy patří k nejbližšímu `if`. Porovnejme výsledky následujících dvou kódů.

```
int a = 5, b = 1, c = 3, foo = 10;

if (a > b){
    if (b > c)
        foo = b;
}
else foo = c;
```

```
int a = 5, b = 1, c = 3, foo = 10;

if (a > b)
    if (b > c)
        foo = b;
    else foo = c;
```

V prvním případě je výsledná hodnota proměnné `foo` rovna 10. Podmínka `a > b` je splněna, ale `b > c` není, druhá podmínka však neobsahuje `else` větev.
V druhém případě `else` větev patří k vnořenému větvení a proměnná `foo` je nastavena na hodnotu proměnné `c`.

Větvení pomocí `switch`

V některých případech je použití větvení pomocí `if` a `else` nepřehledné (je jich potřeba použít mnoho), proto jazyk C nabízí ještě jiné konstrukce umožňující větvení. Jednou z nich je `switch`, která má následující syntaxi.

```
switch (vyraz){
    case konstanta1:
        blok1
    case konstanta2:
```

```
        blok2
    ...
    default:
        blok
}
```

Tato konstrukce se používá pokud v jednotlivých podmínkách kontrolujeme rovnost výrazu s celočíselnou konstantou.
Vyhodnocení tohoto kódu probíhá následovně. Odshora hledáme shodu hodnoty `vyraz` s konstantami `konstanta1`, `konstanta2` a tak dále. S hodnotou `default` se shoduje vždy.[53] Při shodě se začnou provádět příkazy, které následují po shodné konstantě až do té doby, než se narazí na konec `switch`, nebo první příkaz `break`. Ten se většinou píše na konec každého bloku. Konkrétní příklad následuje.

[53] Větev `default` nemusí být uvedena na konci, ale z konvence je vždy jako poslední. Příkazy za `default` se prování vždy až tehdy, když není nalezena žádná vyhovující větev, ať je tato větev uvedena v přepínači kdekoliv.

```
if (a == 1)
    printf("a je jedna");
else if (a == 2)
    printf("a je dva");
else if (a == 3)
    printf("a je tri");
else
    printf("a neni 1, 2 ani 3");
```

Tento kus kódu lze přepsat pomocí konstrukce `switch` následovně.

```
switch (a){
    case 1:
        printf("a je jedna");
        break;
    case 2:
        printf("a je dva");
        break;
    case 3:
        printf("a je tri");
        break;
    default:
        printf("a neni 1, 2 ani 3");
}
```

Pokud má být pro více hodnot proveden stejný kus kódu (blok), je možný následující zápis.

```
switch (vyraz){
    case konstanta1:
    case konstanta2:
    case konstanta3:
```

```
        blok1;
    case konstanta4;
    ...
}
```

Podmínkový operátor

Pro zápis jednoduchého větvení lze použít podmínkový operátor ? :.[54] Obecný zápis jednoduché podmínky pomocí tohoto operátoru vypadá následovně.

[54] Mnohdy se používá jen název *ternární operátor*.

```
podminka ? vyraz1 : vyraz2;
```

Pokud je splněna `podminka`, vyhodnotí se `vyraz1`, pokud ne, vyhodnotí se `vyraz2`. Následující kód

```
if (a < b)
    x = a;
else
    x = b;
```

lze pomocí podmínkového operátoru zapsat následovně.

```
x = (a < b) ? a : b;
```

Cykly

Cykly se používají k opakování části kódu (těla cyklu) vícekrát v závislosti na ukončovací podmínce (kód se vykonává dokud je podmínka splněna) nebo předem daného počtu opakování.
V jazyce C existují tři typy cyklů:

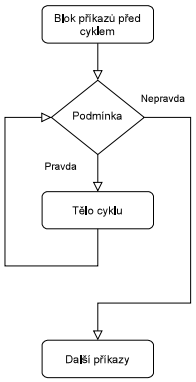
- cyklus `while`,
- cyklus `for`,
- cyklus `do while`.

Každé vykonání části kódu, která se opakuje, se označuje jako *iterace cyklu*.

Cyklus `while`

Postup vykonávání kódu při použití cyklu `while` je zachycen na obrázku 3. Tělo cyklu (blok příkazů) je opakováno, dokud je splněna podmínka.

Obrázek 3: Grafické znázornění průběhu cyklu `while`.



Zápis tohoto cyklu vypadá následovně.

```
while (podminka)
    telo_cyklu
```

`podminka` představuje logický výraz. Pokud je tento výraz vyhodnocen na pravdu, dojde k vykonání bloku kódu `telo_cyklu`.[55] Po jeho vykonání je opět vyhodnocena podmínka a situace se opakuje. Vyhodnocením podmínky na nepravdu program pokračuje kódem za tělem cyklu (říkáme, že *cyklus končí*). Následující část kódu slouží k vypsání čísel od 0 do 9.

[55] Tělo cyklu může být prázdné. Možným použitím je například vynechání všech mezer na vstupu.

```
while (getchar() == ' ')
    ;
```

Pro přehlednost je vhodné psát středník na nový řádek.

```
int j;

j = 0;
while(j < 10){
    printf("%d ",j);
    j = j + 1;
}
```

Nejprve je proměnná `j` inicializovaná na hodnotu `0`. Podmínka `j < 10` je pravdivá, vykoná se tělo cyklu, tedy nejprve se vypíše číslo `j` a následně je hodnota proměnné `j` inkrementována (zvětšena o `1`). Poté se ověří, zda je podmínka splněna. `j` je rovno `1`, takže podmínka je opět pravdivá. Cyklus se opakuje do chvíle, kdy je `j` rovna `10`. Podmínka není splněna, cyklus končí.

Úkol 15
Jak vytvoříte cyklus, který nikdy neskončí?

Cyklus `for`

Druhým typem cyklu je cyklus `for`. Nejčastěji se používá pro cykly s předem daným počtem opakování (iterací). Pro počítání iterací se používá tzv. *krokovací proměnná*. Krokovací proměnné je na začátku cyklu přiřazena počáteční hodnota, která se v každé iteraci zvyšuje (iteruje) a porovnává se s koncovou podmínkou. Cyklus `for` se zapisuje následovně.

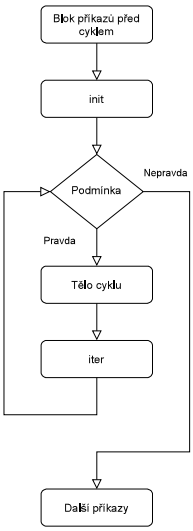
```
for (init; podminka; iter)
    telo-cyklu
```

`init` je blok příkazů oddělených čárkou,[56] které se provedou před samotným cyklem. `podminka` má stejný význam, jako u cyklu `while`. `iter` je seznam příkazů, které jsou odděleny čárkou. Tyto příkazy se vykonají na konci každého vykonání těla cyklu. Průběh vykonávání cyklu je znázorněn na obrázku 4.
Následující kód slouží k výpisu čísel od `0` do `9`.

[56] Jedná se o operátor čárky. `init` je tedy výraz.

```c
int j;

for(j = 0; j < 10; j = j + 1){
    printf("%d ",j);
}
```

Obrázek 4: Grafické znázornění průběhu cyklu `for`.



`init`, `podminka` i `iter` jsou nepovinné, je možné je vynechat (nechat prázdné). Při vynechání `podminka` je podmínka vždy považována za pravdivou.
V následujícím příkladu je uveden stejný cyklus zapsaný různými způsoby.

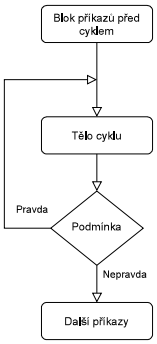
```c
int i = 0;

/* doporucene pouziti cyklu for */
for(i = 0; i < 10; i++){
    printf("%d ",i);
}

/* vyuziti inicializace i pri definici. */
for(; i < 10; i++){
    printf("%d ",i);
}

/* ridici promena se meni v tele cyklu */
for(i = 0; i < 10; ){
    printf("%d ",i++);
}

/* vyuziti operatoru carky */
for(i = 0; i < 10; printf("%d ",i), i++)
    ;
```

Z příkladu je patrné, že poslední tři způsoby zápisu nejsou tak přehledné, jako první.

Cyklus `do while`

Posledním typem cyklu je cyklus `do while`, jehož průběh vykonávání je znázorněn na obrázku 5. Konstrukce je analogická cyklu `while`, pouze s tím rozdílem, že podmínku testujeme poté, co je vykonáno tělo cyklu. To znamená, že u `do while` vždy proběhne tělo cyklu alespoň jednou. Tento cyklus se zapisuje následovně.

Obrázek 5: Grafické znázornění průběhu cyklu `do while`.




```c
do
   telo-cyklu
while (podminka);
```

Příkazy `break` a `continue`

Při práci s cykly se nám mohou hodit speciální příkazy a to příkazy `break` a `continue`.
Příkaz `break` způsobí okamžité ukončení vykonávání cyklu a program pokračuje až za ním.
Příkaz `continue` způsobí okamžité ukončení vykonávání těla, ale nedojde k ukončení celého cyklu. Pokračuje se další iterací (je znovu zkontrolována podmínka a při její platnosti se tělo cyklu vykonává znovu).
Pro představu, jak tyto příkazy pracují porovnejte následující kódy.

```
int j;
for(j = 1; j < 10; j = j + 1){
    if((j % 3) == 0)
        continue;
    printf("%d ", j);
}
```

Kód vypíše čísla 1, 2, 4, 5, 7, 8. Použití příkazu `continue` způsobí to, že pro čísla dělitelní 3 se neprovede příkaz `printf`.

```
int j;
for(j = 1; j < 10; j = j + 1){
    if((j % 3) == 0)
        break;
    printf("%d ", j);
}
```

Výstupem tohoto kódu budou čísla 1, 2. V iteraci, ve které je `j` rovno 3, dojde k vykonání příkazu `break`, jenž způsobí úplné ukončení cyklu.

Příkaz `goto`

Málo používaný příkaz měnící tok programu je příkaz `goto`. Ten má následující syntaxi.

```
goto navesti;

...

navesti:

...
```

Pokud program narazí při běhu na tento příkaz, pokračuje se vykonáváním programu za *návěštím* `navesti`. Návěští může být téměř

kdekoliv. Není možné však „skákat“ mezi funkcemi.
V dobře napsaných programech se tento příkaz téměř nevyskytuje, protože v jazyce C se mu lze vždy vyhnout. Jedno z mála odůvodnitelných (možná jediné) použití tohoto příkazu je opuštění vnořených cyklů (příkazem `break` opustíme jen aktuální cyklus). Viz následující příklad.

```
for (i = 0; i < 10; i++){
    for (j = 0; j < 10; j++){
        for (k = 0; k < 10; k++){
            /* x je pole a x[k] vraci jeho k-ty prvek */
            if (x[k] == 0)
                goto chyba;
            printf("%d", (x[i] + x[j]) / x[k]);
        }
    }
}
...
chyba:
printf("Nelze delit nulou\n");
```

Úkol 16
Přepište příklad bez použití příkazu `goto`.

Za žádných okolností by neměl být používán příkaz `goto` pro skok dovnitř bloků větví příkazu `if` z vnějšku, skok z kladné větve `if` do negativní, skok dovnitř těla `switch` z vnějšku, skok dovnitř složeného příkazu (bloku) z vnějšku.

Příkaz `return`

Posledním příkazem, který si v této kapitole představíme je příkaz `return`. Ten způsobí ukončení právě vykonávané funkce (v případě funkce `main()` to znamená, že se ukončí celý program).[57] Pomocí `return` se z funkce vrací nějaká hodnota. Ta by měla odpovídat návratové hodnotě funkce. O tom ale budeme podrobněji mluvit později. Pokud by příklad uvedený v sekci `goto` neobsahoval jiný kód než cyklus, bylo by vhodnější nahradit `goto` příkazem `return`.[58]

```
for (i = 0; i < 10; i++){
    for (j = 0; j < 10; j++){
        for (k = 0; k < 10; k++){
            /* x je pole a x[k] vraci jeho k-ty prvek */
            if (x[k] == 0){
                printf("Nelze delit nulou\n");
                return 1;
            }
            printf("%d", (x[i] + x[j]) / x[k]);
        }
    }
}
```

[57] Ve standardní knihovně `stdlib` je definována funkce `exit()`, která vyvolá ukončení programu ať je zavolána z jakékoliv funkce, ne jen `main()`.

[58] V příkladech jsme doposud používali ve funkci `main()` příkaz:

```
return 0;
```

Značí to, že z funkce `main()` vracíme hodnotu `0`. To znamená, že program skončil úspěšně. Při vrácení jiné hodnoty dáváme systému informaci, že při vykonávání kódu nastal nějaký problém.
K výpisu toho, s jakou hodnotou program skončil, můžeme použít v terminálu (konzoli) příkaz
`echo $?`
v případě, že pracujeme na UNIX systému.
`echo %ERRORLEVEL%`
v případě, že pracujeme na Windows.

Cvičení

Úkol 17
Za použití podmínkového operátoru napište program, který pro zadané číslo vypíše jeho absolutní hodnotu.

Úkol 18
Napište program, který rozhodne, zda zadaný rok je přestupný. (Definici přestupného roku naleznete například na wikipedii `https://cs.wikipedia.org/wiki/P%C5%99estupn%C3%BD_rok`).

Úkol 19
Napište program, který rozhodne, zda je zadané písmeno malé nebo velké.

Úkol 20
Pro zadané číslo od 1 do 10, vypište sestupnou posloupnost čísel od zadaného čísla po 1. (pro 4 bude výstup 4 3 2 1). K řešení zkuste vhodně použít konstrukci `switch`.[59]

[59] Vhodnější je použít cyklus. Tento úkol je spíše k zamyšlení.

Úkol 21
Napište program, který načte celá čísla a a b a pak

1. vypíše prvních a násobků čísla b
2. spočítá a-tou mocninu čísla b
3. určí kolik číslic má číslo a
4. vypočítá a-té Fibonacciho číslo
5. sečte všechna čísla větší než a a menší než b

Úkol 22
Napište program, který pro zadané číslo vrátí číslo zapsané pozpátku. (Pro 1234 vrátí číslo 4321)

Úkol 23
Pro zadané *n* vykreslete do konzole následující obrázky

1. pro n=3

```
      *
    * * *
  * * * * *
```

   pro n = 4

```
        *
      * * *
    * * * * *
  * * * * * * *
```

2. pro n=2

```
    *
  * * *
    *
```

   pro n = 3

```
      *
    * * *
  * * * * *
    * * *
      *
```

3. šachovnici o straně *n* pro n = 4

```
. * . *
* . * .
. * . *
* . * .
```

Úkol 24
Přiřaďte výstupy k blokům kódu po dosazení bloku do následujícího kódu.

```
#include <stdio.h>

int main(){
    int x = 0;
    int y = 0;
    while (x < 5){
        /* sem
           vlozte
           blok
           kodu */
        printf("%d%d ", x, y);
        x = x + 1;
    }
    return 0;
}
```

Bloky kódu:

```
y = x - y;
```

```
y = y + x;
```

```
y = y + 2;
if (y > 4)
    y = y - 1;
```

```
x = x + 1;
y = y + x;
```

```
if (y < 5){
    x = x + 1;
    if (y < 3)
        x = x - 1;
}
y = y + 2;
```

Možné výstupy:[60]

```
22 46
```

```
11 34 59
```

```
02 14 26 38
```

```
02 14 36 48
```

```
00 11 21 32 42
```

```
11 21 32 42 53
```

```
00 11 23 36 410
```

```
02 14 25 36 47
```

Řešení ověřte spuštěním kódu.