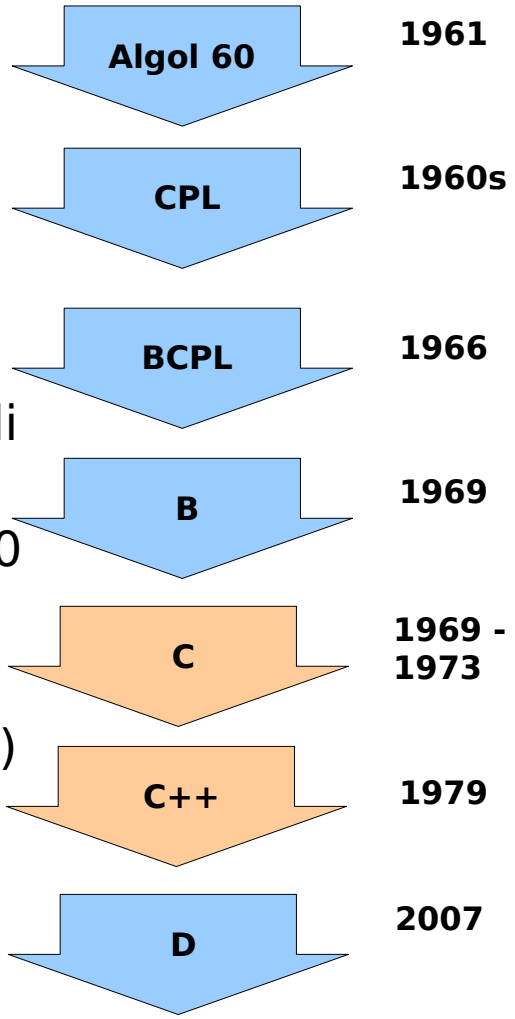
Programování v jazyce C pro chemiky
(C2160)

1. Úvod do C, příkaz podmínky

Historie jazyka C

- Jazyk C vyvinul Dennis Ritchie v AT&T Bell Labs v letech 1969–1973
- Počátek vývoje C úzce spjat s počátky vývoje Unixu – zpočátku byl Unix napsaný v assembleru, ale v roce 1973 byl přepsán do C
- 1978 – Brian Kernighan (AW**K**) a Dennis Ritchie publikovali knihu "The C Programming Language" (**K&R C**)
- 1989–90 – standard ANSI X3.159-1989, ISO/IEC 9899:1990 – **C89, C90, ISO C, ANSI C**
- 1999 – ISO/IEC 9899:1999 – **C99**
- 2011 – ISO/IEC 9899:2011 – **C11**, 2017 oprava chyb (**C17**)



Zkratky:

ANSI – American National Standards Institute
ISO – International Organization for Standardization
IEC – International Electrotechnical Commission

Jazyk C

- Jazyk C je široce používaný pro tvorbu operačních systémů, aplikačních programů, počítačových her, programování elektronických zařízení
- Je široce využíván ve vědeckých oborech pro tvorbu výpočetně náročných programů (společně s jazykem Fortran)
- Ze syntaxe jazyka C vycházejí C++, Java, C#, JavaScript, PHP, C shell (csh, tcsh)

Výhody:

- Vysoká výpočetní rychlost a paměťová efektivita programů
- Možnost nízkoúrovňových operací pro přímou komunikaci s hardwarem (operační systémy, ovladače hardwaru)
- Široká dostupnost kvalitních překladačů a knihoven

Nevýhody:

- Minimum omezení, jazyk nechává programátora dělat cokoli, bez ohledu na to, zda je to moudré – nepříliš vhodný jako první programovací jazyk
- Nutnost kompilace zdrojového kódu po každé změně v programu

Kompilace programu v C

- Jazyk C patří mezi takzvané **kompilované jazyky**, to znamená, že po vytvoření souboru se zdrojovým kódem programu v C musíme spustit **překladač** (kompilátor), který vygeneruje soubor se spustitelným kódem
- Pod Linuxem je standardně dostupný kompilátor ***gcc***, kompilaci spustíme následujícím způsobem:

gcc -o spustitelny_soubor zdrojovy_soubor.c

Zdrojový soubor obsahující kód programu v jazyce C vytvoříme v textovém editoru (soubory mají obvykle koncovku ***.c***)

Spustíme kompilátor který ze zdrojového souboru **.c* vygeneruje spustitelný soubor

Spustitelný soubor obsahuje instrukce pro procesor a lze ho spustit např. z příkazového řádku

Programátorský editor Kate

- Editor *Kate* je programátorský editor který umožňuje automatické odsazování textu, barevné označení klíčových slov jazyka, vyznačení párování závorek a pod.
- Pro správné odsazování textu odpovídající jazyku C je potřeba, aby editovaný soubor měl koncovku ***.c***
- Kontrola párování závorek: pokud umístíme textový kurzor na složenou nebo obyčejnou závorku, označí se příslušná párová závorka (tučně a případně žlutou barvou)
- Multiplatformní, dostupný ve Windows Store

Kompilace programu v C - příklad

```
#include <stdio.h>

int main()
{
   printf("Hello, World!\n");
   return 0;
}
```

- Vytvořte složku, do které budete ukládat programy ze cvičení
- Spusťte editor *kate*, napište výše uvedený program v jazyku C a uložte ho do souboru *helloworld.c*
- V okně terminálu přejděte do složky s vytvořeným souborem *helloworld.c* a spusťte kompilátor: *gcc -o helloworld helloworld.c*
- Kompilátor vygeneruje spustitelný soubor *helloworld*
- V okně terminálu spusťte program: *./helloworld*

Chybové hlášky kompilátoru

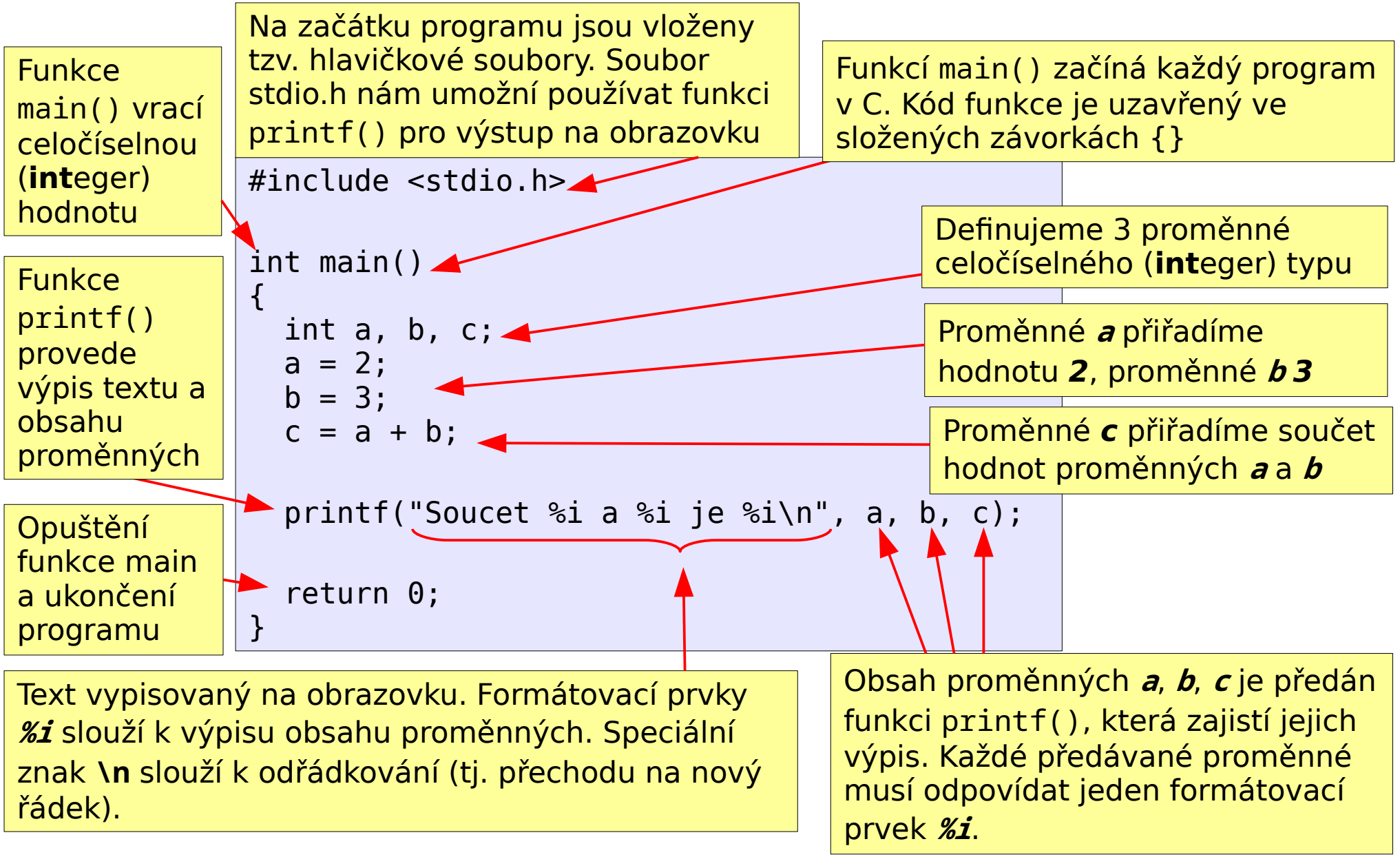
```
#include <stdio.h>

int main()
{
   printf("Hello, World!\n"));
   return 0;
}
```

- Za funkci `printf` přidáme závorku navíc
- Při překladu nahlásí kompilátor chybu
- Uveden je popis chyby a číslo řádku
- POZOR! – číslo řádku nemusí vždy odpovídat skutečné pozici chyby, ale označuje místo, od kterého překlad nedává kompilátoru smysl. Skutečná chyba se může nacházet na některém z předešlých řádků.
- Příklad: vynechejte středník za funkcí `printf` – chyba bude nahlášena na následujícím řádku
- Čísla řádků lze v *Kate* zobrazit: Menu: *View / Show Line Numbers*

Struktura programu v C

Program pro sčítání dvou celých čísel:



Struktura programu v jazyce C

- Veškeré příkazy jazyka C jsou zapsány ve funkcích
- Program začíná vždy funkcí `main()`, tuto funkci tedy musí nezbytně obsahovat všechny programy napsané v jazyce C
- Kód funkce je uzavřen ve složených závorkách `{}`
- Každý příkaz je zakončen středníkem
- Jazyk C je case-sensitive, tj. rozlišuje velká a malá písmena (např. proměnná `N` vůbec nesouvisí s proměnnou `n`)

Komentáře v C

- 1. typ: použití dvou lomítek, vše za nimi až do konce řádku je komentář (**C99**): *// Vše za dvěma lomítky je komentář*
- 2. typ: lze použít přes více řádků: */* Nějaký text komentáře */*

```
// Komentar muze byt i mimo funkci

int main()
{
  /* Samozrejme muze byt i ve funkci. Tento typ komentare
     s lomitkem a hvezdickou muze pokracovat i pres vice radku */
  int a, /* Dokonce i tady muze byt */ b, c;
  a = 2;  // Vse za dvema lomitky az do konce radku je komentar
  b = 3;  //Neni treba delat za lomitky mezeru, ale je to prehlednejsi
  c = a + b;
  //c = a – b; komentare lze pouzit i k docasnemu zneplatneni kodu

  printf("Soucet %i a %i je %i\n", a, b, c);
  /* Pokud je komentar viceradkovy, dava se
   * nekdy na zacatek vsech radku hvezdicka.
   * Neni to sice nutne, ale znacne to zprehlednuje
   * text komentare, hlavne pokud je hodne dlouhy.
   */

  return 0;
}
```

Mezery v C

- Většinu mezer lze vynechat nebo libovolně přidávat, hlavně před a po znacích =+-*/[](){}.,;:?<>&|~%"
- Mezery nelze vkládat doprostřed názvů (např. funkcí, proměnných, typů apod.)
- Mezera je nezbytná k oddělení jména typu od jména proměnné nebo funkce a také za příkazem `return`
- Z důvodů přehlednosti je vhodné mezery hojně používat

```
int␣main␣(␣)
{
  int␣a,␣b,␣count␣;
  a␣=␣␣␣2␣;
  b␣=␣3;
  c␣=␣a␣+␣b;

  printf␣(␣"Soucet %i a %i je %i\n",␣a,␣b␣,␣c␣)␣;

  return␣0;
}
```

␣ mezery povinné
␣ mezery nepovinné

Proměnné

- Každá proměnná musí být před prvním použitím definována tj. uveden typ a její jméno
- Definici proměnné je vhodné uvádět na začátku funkce
- Základní typy proměnných:
  - **int** - celá čísla (**int**eger)
  - **float** – desetinná čísla (**float**ing point number)
  - **char** – znak (**char**acter)
- Při definici uvedeme jméno typu, mezeru a seznam proměnných oddělených čárkou
- Počáteční hodnotu proměnné lze nastavit pomocí **=** (mluvíme o tzv. inicializaci proměnné)

```
int a, b, c;    // definujeme tri celociselne promenne a, b, c
int d = 5;      // definujeme promennou d a inicializujeme ji hodnotou 5
float m, p;     // definujeme necelociselne promenne m a p
float g = 1.75; // definujeme promennou g a inicializujeme ji hodnotou 1.75
char z;         // definujeme znakovou promennou z

int teplota, vlhkost;  // jmeno promenne muze byt i delsi nez jeden znak
int partial_charge3; // muzeme pouzivat pismena, cislice, podtrzitko, ne
                     // vsak mezeru
int MaxValue;      // lze pouzit i velka pismena
int maxValue;      // toto je jiz jina promenna (jazyk C je case-sensitive!)
```

Konstanty

- Konstanty jsou číselné nebo znakové hodnoty, které používáme např. pro přiřazení hodnot proměnným
- Nejpoužívanější jsou následující konstanty:
  - Celočíselné konstanty zapisujeme jako běžná čísla (např. 1, -4, 178)
  - Reálné konstanty zapisujeme s desetinnou tečkou (1.14, -58.146)
  - Znakové konstanty zapisujeme jako **jeden znak** uvedený **v apostrofech** (např. `'A'`, `'r'`)
- Konstanty používáme např. pro inicializaci proměnných; pro každou proměnnou používáme odpovídající typ konstanty (tj. pro reálnou proměnnou použijeme reálnou konstantu, pro znakovou proměnnou použijeme znakovou konstantu atd.)
- Všechny proměnné bychom měli vždy inicializovat vhodnou implicitní hodnotou (často 0)

```
int d = 5;    // inicializace celociselne promenne celociselnou konstantou
int a = 0, b = 0;
float g = 1.75; // inicializace realne promenne realnou konstantou
float m = 0.0, n = 0.0;
char z = 'D';   // inicializace znakove promenne znakovou konstantou
```

Přiřazovací příkaz a aritmetické operátory

- Přiřazovací příkaz **=**
- Aritmetické operátory **+ - * /**
- Operátor **%** poskytuje zbytek po dělení celých čísel (pozor, je-li dělenec záporný, je zbytek též záporný)
- Pro určení priority operací lze používat závorky

```
int a, b, c, d;  // definice promennych a, b, c
a = 5;           // promenne a priradi hodnotu 5
b = 7;           // promenne b priradi hodnotu 7
c = 13;          // promenne c priradi hodnotu 13

d = a + b;       // promenne d priradi hodnota souctu a plus b
d = a - b;
d = a * b;
d = a / b;       // POZOR!: deleni nulou zpusobi okamzite ukonceni programu
d = c % a;       // Zbytek po deleni cisla 13 cislem 5 coz je 3

d = a + b / c;   // Provede se deleni b/c, pak se pricte a
d = (a + b) / c; // Secte se a plus b a pak se to vydeli c
```

Funkce `printf()` - výstup textu na obrazovku

- Funkce `printf()` slouží k výpisu textu na obrazovku (terminál)
- Prvním parametrem funkce je text, který bude vypsán na obrazovku (zapisuje se v uvozovkách)
- Dále může následovat seznam libovolného počtu proměnných jejichž hodnoty mají být vypsány
- Místo v textu, kde se vypíše hodnota proměnných, se specifikuje pomocí formátovacích prvků:
  - **`%i`** pro proměnné typu `int` (lze i podobně fungující **`%d`**)
  - **`%f`** pro proměnné typu `float`
  - **`%c`** pro proměnné typu `char`
- Speciální znak **`\n`** způsobí přechod na nový řádek

```
int a = 3, b = 6, c = 9;
float m = 2.71, n = 14.3;
char c1 = 'D', c2 = 'j';

printf("Jednoduchy text\n");
printf("Cislo a ma hodnotu %i zatimco b je %i\n", a, b);
printf("Vypis celeho cisla %i a desetinneho cisla %f\n", c, m);
printf("Vypis dvou znaku za sebou: %c%c\n", c1, c2);
```

Použití matematických funkcí

- Pro zpřístupnění matematických funkcí přidáme na začátek programu: `#include <math.h>`
- Při kompilaci je potřeba přidat matematickou knihovnu pomocí **`-lm`** (`l` jako **l**ibrary a `m` jako **m**ath), např:
  `gcc -o mujprogram mujprogram.c -lm`
- Nejpoužívanější matematické funkce:
  `sqrt(x)` – odmocnina z x
  `pow(x, y)` – x umocněno na y (u druhé, třetí mocniny však dáváme přednost jednoduchému násobení)
  `exp(x)` – *e* na x-tou
  `log(x)` – přirozený logaritmus x
  `cos(x)` – kosinus x, **`sin(x)`** – sinus x (úhel v radiánech)

```
int n = 3;
float a = 7.3, b = 2.5, result = 0.0;

result = sqrt(a);
result = pow(a, n) + 7.4;
result = a + cos(b);
result = (sqrt(a) + sqrt(b)) / 2;
```

Funkce scanf() - čtení vstupu z klávesnice

- Funkce `scanf()` slouží k získání vstupu z klávesnice
- Prvním parametrem funkce je formátovací text, který obsahuje jeden nebo více formátovacích prvků
- Do formátovacího textu **nezadáváme žádný text**, používáme jen formátovací prvky
- Pokud použijeme více než jeden formátovací prvek, oddělujeme je zpravidla mezerou – mezeru pak musíme použít pro oddělení hodnot zadávaných z klávesnice
- Před načítáním hodnot pomocí scanf() je zpravidla vhodné vypsat na obrazovku výzvu k zadání příslušných hodnot, proto nejdříve zavoláme printf(), která text vypíše, a pak načítáme pomocí scanf()
- Za formátovací text funkce `scanf()` zapíšeme tolik proměnných, kolik bylo použito formátovacích prvků
- **POZOR**: Před název proměnné musíme vložit znak **&**

```
int a = 0;
float m = 0.0, n = 0.0;

printf("Zadej cele cislo a stiskni Enter: ");
scanf("%i", &a);

printf("Zadej dve desetinna cisla oddelena mezerou: ");
scanf("%f %f", &m, &n);
```

Příklad použití funkce scanf()

Program demonstrující čtení vstupu z klávesnice:

- Program si vyžádá dvě desetinná čísla A a B a vypíše jejich součet

```
int main()
{
  float a = 0.0, b = 0.0, c = 0.0;

  printf("Zadej A a B oddelene mezerou a stiskni Enter: ");
  scanf("%f %f", &a, &b);

  c = a + b;

  printf("Soucet %f a %f je %f\n", a, b, c);

  return 0;
}
```

Příkaz podmínky `if`

- Příkaz slouží k podmíněnému provádění příkazů
- Zapisuje se:

  if (*podmínka*) *příkaz*;
- Můžeme specifikovat i příkaz který se povede při nesplnění podmínky:

  if (*podmínka*) *příkaz1*; **else** *příkaz2*;

```
int a = 4, b = 0;

if (a > 3)
  b = 10;     // Tento prikaz se provede pouze pokud a > 3


if (a > 8)
  b = 10;     // Tento prikaz se provede pouze pokud a > 8
else
  b = 0;      // Tento prikaz se provede pouze pokud neni a > 8
```

Příkaz podmínky `if`

- Pokud potřebujeme podmíněně vykonat více než jeden příkaz, použijeme složené závorky

```
int a = 4, b = 0, c = 0, d = 0;

if (a > 3)
{
  b = 10;
  c = a + b;
}

if (a > 8)
{
  b = 5;
  c = a - b;
}
else
{
  b = 7;
  d = a + b
}
```

Příkaz podmínky `if`

- Složené závorky je používat i pro jeden příkaz, hlavně je-li přítomen *else* blok nebo jde o vnořenou podmínku
- Bez složených závorek může přidání nového příkazu do těla if/else úplně změnit význam podmínek

```
int a = 4, b = 1;

if (a > 3)
  if (a > 8)
    b = 3;
else
  b = 2;
/* Jakou hodnotu ma b v tomto miste? */

if (a > 3)
{
  if (a > 8)
  {
    b = 3;
  }
  else
  {
    b = 2;
  }
}
```

Příkaz podmínky `if`

- Závorky a „else“ můžeme umístit dle vkusu, hlavně konzistentně!
- Uzavírací složená závorka by ale vždy měla být na stejné úrovni odsazení, jako klíčové slovo `if`

```
if (a > 3)
{
  if (a > 8)
  {
    b = 3;
  }
  else
  {
    b = 2;
  }
}

if (a > 3) {
  if (a > 8) {
    b = 3;
  } else {
    b = 2;
  }
}
```

Příkaz podmínky `if` – konstrukce podmínky

- Pro konstrukci podmínky používáme následující operátory:

==	rovnost
!=	nerovnost
<	menší
<=	menší nebo rovno
>	větší
>=	větší nebo rovno
||	nebo (logické OR)
&&	a zároveň (logické AND)

- Operátory se vyhodnocují podle priority
- Zjednodušená tabulka priorit operátorů:

< <= > >=	nejvyšší priorita
== !=	
&&	
||	nejnižší priorita

- Prioritu lze ovlivnit pomocí závorek
- Doporučení: prioritu raději přesně specifikovat pomocí závorek (menší pravděpodobnost, že uděláme chybu)

Příkaz podmínky `if` – konstrukce podmínky

Následující ukázka je zjednodušená, příkazy za podmínkami nejsou uvedeny

```
int a = 4, b = 5, c = 9;
char ch1 = 'A', ch2 = 'D';

if (a == 3)                  // Jednoducha podminka
if (a != 3)                  // Jednoducha podminka
if (b <= 3)                  // Jednoducha podminka
if (ch1 == 'B')              // Jednoducha podminka
if (ch1 != 'h')              // Jednoducha podminka

if (a > 3 && b <= c)         // Trochu slozitejsi podminka
if ((a > 3) && (b <= c))     // Tataz podmínka, ale s pouzitim zavorek

if (a > 3 || b != c && b <= c)   // Prioritu ma && pred ||
if (a > 3 || (b != c && b <= c)) // Totez s pouzitim zavorek
if ((a > 3 || b != c) && b <= c) // Tady zavorky zmenily poradi vyhodnoceni
```

Příklad použití podmínky

Program načte z klávesnice celá čísla a, b a vypíše, které je větší

```
int main()
{
  int a = 0, b = 0;

  printf("Zadej A a B oddelene mezerou a stiskni Enter: ");
  scanf("%i %i", &a, &b);

  if (a == b)
  {
    printf("Vysledek: A je rovno B\n");
  }
  else
  {
    printf("Vysledek: A je ruzne od B\n");
  }

  return 0;
}
```

Dodržujte následující pravidla

- Všechny **proměnné vždy inicializujte** vhodnou implicitní hodnotou při její definici (nejčastěji 0)
- Dbejte na **správné odsazování textu**. K odsazování by mělo docházet v editoru *Kate* automaticky, je ale třeba soubor hned po vytvoření uložit s koncovkou `.c`. Není-li text správně zarovnán, označte veškerý text a pak aplikujte Menu: Tools/Align. Pokud nedojde ke správnému zarovnání, příčinou může být špatná koncovka souboru (musí být *.c*), nesprávně spárované závorky nebo chybějící středník na konci příkazu.
- **Úlohy odevzdávejte do odevzdávárny** předmětu C2160. Do odevzdávárny „Cvičení 1“ vkládejte soubory jednotlivých úloh pojmenované tak, aby jejich jméno končilo číslicí označující úlohu. Není třeba do názvu souboru vkládat vaše jméno, to doplní IS sám. **Odevzdávejte pouze soubory s kódem v C**, nikoliv spustitelné soubory.

Úlohy

1. Vytvořte program pro řešení kořenů kvadratické rovnice. Program si od uživatele vyžádá zadání reálných koeficientů *a*, *b*, *c*, spočítá kořeny a vypíše je na obrazovku. Ošetřete situaci, kdy je diskriminant záporný nebo koeficient *a* je roven 0. **1 bod**
2. Vytvořte program pro sčítání reálných čísel. Čísla budou zadávána z klávesnice. Program se na začátku zeptá, kolik čísel budete chtít sčítat (max. pět). Potom si vyžádá zadání čísel z klávesnice. Nakonec zobrazí výsledek. **1 bod**
3. Vytvořte program který vypíše seznam možných operací (sčítání, odčítání, násobení, dělení) označených čísly 1 až 4. Uživatel si vybere, kterou operaci chce provést. Potom bude vyzván k zadání dvou reálných čísel. Nakonec se vypíše výsledek. **1 bod**
4. Modifikujte předchozí program tak, že volba operace se neprovede pomocí čísla 1 až 4, ale zadáním jednoho ze znaků +, -, *, /. **nepovinná, 1 bod**