**Programování v jazyce C pro chemiky**
(C2160)

# 2. Příkazy cyklu `while` a `do`, ladění programů

# Příkaz cyklu `while`

- Příkaz cyklu **while** slouží k opakovanému provádění příkazů
- Zapisuje se:

  **while** (*podmínka*) {
  *příkazy*;
  }
- Příkazy se budou vykonávat opakovaně tak dlouho, dokud je splněna podmínka (ta se testuje před každým průchodem)

```
int a = 0, b = 3, c = 1;

while (a < 5) {
  c = c * b;
  a = a + 1;
}
```

# Příkaz cyklu do

- Cyklus **do** funguje podobně jako **while**, ale podmínka je uvedena na konci
- Zapisuje se:

  **do** {
  *příkazy*;
  } **while** (*podmínka*);

- Na rozdíl od cyklu **while** při použití cyklu **do** vždy dojde minimálně k jednomu průchodu cyklem (podmínka se vyhodnocuje až po průchodu)
- Na konci cyklu je za podmínkou vždy středník

```
int a = 0, b = 3, c = 1;

do {
  c = c * b;
  a = a + 1;
} while (a < 5);    // Nezapominat na strednik na konci
```

# Příkaz break

- Příkaz **break** slouží k vyskočení z cyklu
- Po vykonání příkazu **break** program pokračuje kódem za cyklem

```
int a = 0, b = 3, c = 1;

while (a < 5) {
  c = c * b;
  if (c > 100)
    break;     // Zde se vyskoci z cyklu (pokud c > 100)
  a = a + 1;
}

a = b + c;     // Tady program pokracuje pokud byl cyklus ukoncen
               // nebo pokud z nej bylo vyskoceno pomoci break
```

# Příkaz break

- V případě vnořených cyklů příkaz **break** ukončí jen ten nejvnitřnější, v němž se přímo nachází

```
int a = 0, b = 2, c = 1;

while (a < 5) {
  b = 2;
  while (b < 4) {
    c = c * b;
    if (c > 100)
      break;    // Zde se vyskoci z vnitrniho cyklu
    b = b + 1;
  }
  a = a + 1;    /* Tento prikaz se provede celkem petkrat,  *
                 * vzdy po skonceni vnitrniho cyklu         *
                 * (at uz kvuli breaku nebo podmince)       */
}
```

# Kompilace programů přes Kate

- Editor Kate dokáže přímo spouštět kompilátor, aniž bychom to museli dělat ručně v terminálu
- Výhodou je, že Kate zpracuje případná chybová hlášení a naváže je na dotčená místa v kódu

- Zapnutí podpory: Settings→Configure Kate→Plugins→Build Plugin
- Nastavení: V panelu Build Output na záložce Target Settings vytvoříme novou sadu cílů (tlačítkem Create new set of targets), pokud v seznamu žádná není
- Potom můžeme dvojklikem do pole vpravo na řádku Build nastavit požadovaný příkaz pro kompilaci, například `gcc -o "%n" "%f"` (za "%f" bude automaticky dosazena cesta k aktuálnímu souboru, "%n" je totéž bez přípony)
- Řádek Build může být třeba povolit zaškrtnutím políčka vlevo
- Ostatní řádky (Clean, Configure) můžeme smazat (tlačítkem Delete current target)

- Nakonec tlačítkem Build selected target spustíme kompilaci

# Použití debuggeru GDB

- Ladění složitějších programů si můžeme usnadnit použitím debuggeru
- Kompilátoru předáme navíc argument **`-g`**
- Debugger spustíme příkazem **`gdb ./nazev_programu`**
- Potom můžeme debugger ovládat následujícími příkazy (lze zkracovat, v závorce je plné znění příkazu):

  `run` *`argumenty…`* – spustí laděný program

  `break` *`jmeno_funkce`* – definuje breakpoint (zastaví program na dané funkci)

  `bt (backtrace)` – vypíše řetězec volaných funkcí k aktuálnímu místu

  `cont (continue)` – pokračuje v běhu programu po přerušení

  `n (next)` – provede aktuální řádek programu a zase zastaví

  `l (list)` – vypíše zdrojový kód kolem aktuálního místa

  `p` *`jmeno_promenne`* `(print)` – vypíše hodnotu proměnné

  `kill` – ukončí laděný program

  `quit` – ukončí celý debugger

# Integrace GDB do editoru Kate

- Debugger GDB můžeme ovládat i grafickou cestou přes Kate
- Zapnutí integrace: Settings→Configure Kate→Plugins→GDB
- Nastavení: V panelu Debug View na záložce Settings vytvoříme "target" (sada nastavení pro jeden laděný program) a nastavíme název spustitelného souboru, pracovní adresář a argumenty
  - Vyžaduje-li program vstup od uživatele, zapneme Redirect IO (vstup pak píšeme do záložky IO)
  - Nevidíme-li v panelu nástrojů ladicí tlačítka (Step, Continue), musíme panel zapnout přes Settings→Toolbars Shown→GDB Plugin

- Ladění pak spustíme pomocí Debug→Start Debugging
- Dále krokujeme pomocí Step In (vstupuje do volané funkce) / Over, nastavujeme zarážky pomocí Toggle Breakpoint, Continue pokračuje v běhu k další zarážce
- V panelu Locals and Stack vidíme hodnoty proměnných a zásobník volání
- Ladění ukončíme přes Debug→Kill

# Dodržujte následující pravidla

- Na začátek programu nezapomeňte vložit hlavičkové soubory:
  #include <stdio.h> (pokud používáte funkce printf() a scanf())
  #include <math.h> (používáte-li matematické funkce sqrt(), ...)
  Při použití matematických funkcí je navíc potřeba použít parametr -lm (l je zde malé L), aby došlo k připojení matematické knihovny
- Všechny proměnné vždy inicializujte vhodnou implicitní hodnotou při její definici.
- Za definicí každé z proměnných uveďte krátký komentář popisující, k čemu proměnná slouží (není-li to naprosto zjevné z jejího názvu).
- Nezapomeňte, že v podmínkách `if` a v podmínkách cyklů musíme při porovnávání proměnných používat dvě **==**, nikoliv jedno =.
- Ve funkci `scanf()` nezapomeňte před název proměnné přidat **&**.
  Při volání funkce `printf()` se naopak tento znak nepoužívá.
- Dbejte na správné (konzistentní a přehledné) odsazování textu. Při správně nastaveném editoru by mělo ke správnému odsazování textu docházet automaticky.

# Úlohy

1. Vytvořte program, který vypíše čísla od 1 po hodnotu zadanou uživatelem. **1 bod**
2. Vytvořte program, který si vyžádá od uživatele celé kladné číslo a určí, jestli se jedná o prvočíslo. **1 bod**
3. Vytvořte program, který vypíše prvních *n* prvočísel. Číslo *n* bude zadáno uživatelem po spuštění programu. **nepovinná, 2 body**
4. Program z úlohy 3 upravte tak, aby vypisoval prvočísla oddělená čárkou, vždy deset prvočísel na každý řádek. **nepovinná, 1 bod**