Programování v jazyce C pro chemiky
(C2160)

10. Kompilace složitějších programů, grafická knihovna g2

Preprocesor jazyka C

- Zdrojový text programu je před kompilací zpracován preprocesorem, který provede vložení externích souborů, vynechání částí kódu, substituci vybraných slov, odstranění komentářů atd.
- Soubor se po zpracování preprocesorem nikde neukládá, ale je přímo předán překladači jazyka C
- Chceme-li spustit preprocesor samostatně a prohlédnout si zpracovaný soubor, spustíme kompilátor s parametrem **-E**

  gcc -E vstupni_soubor -o vystupni_soubor (viz *man gcc*)

  nebo spustíme přímo preprocesor:

  cpp vstupni_soubor vystupni_soubor (viz *man cpp*)
- Činnost preprocesoru je řízena direktivami preprocesoru, každá direktiva preprocesoru začíná znakem **#**, direktivy mají zcela odlišnou syntaxi než příkazy jazyka C (např. se neukončují středníkem)
- Další informace:

  https://en.wikipedia.org/wiki/C_preprocessor

  https://www.cprogramming.com/tutorial/cpreprocessor.html

Direktiva vložení souboru `#include`

- Direktiva `#include` slouží k vložení souboru
- Jméno souboru píšeme v lomených závorkách nebo v uvozovkách:

```
#include  <stdio.h>
#include  "../include/prog.h"
```

- Je-li jméno souboru uvedeno v lomených závorkách, je vkládaný soubor hledán ve standardních systémových adresářích (zpravidla */usr/include*)
- Je-li jméno souboru uvedeno v uvozovkách, hledá se soubor v adresáři, ve kterém se nachází volající soubor
- Pokud specifikujeme celou cestu k souboru, nezáleží na tom, kterou metodu zápisu použijeme (obvyklé jsou uvozovky)
- Direktivu `#include` používáme nejčastěji pro vkládání tzv. hlavičkových souborů, které obsahují deklarace funkcí, typů a globálních proměnných
- Hlavičkové soubory mívají obvykle koncovku *.h*

Direktiva definující makro #define

- Direktiva #define slouží k definování tzv. makra

  #define JMENO_MAKRA libovolny text rozvoje

- Direktiva způsobí, že preprocesor nahradí každý výskyt slova JMENO_MAKRA textem libovolny text rozvoje
- Nahrazovaný text může být libovolně dlouhý včetně mezer; pokud pokračuje na dalším řádku musí být na konci řádku zpětné lomítko

  #define DLOUHE_MAKRO Tady je nejaky text, který \
  se nevejde na jeden radek

- Substituce definovaná makrem se začne uplatňovat až od řádku na němž je makro definováno
- Platnost makra lze zrušit direktivou #undef, od toho okamžiku nebude výskyt JMENO_MAKRA ničím substituován

  #undef JMENO_MAKRA

- Direktivy #define zpravidla zapisujeme na začátek programu a jejich platnost trvá od daného místa až do konce souboru

Symbolické konstanty

- Direktiva `#define` se často používá ke specifikaci tzv. symbolických konstant, kdy nahrazovaným textem je číslo, znak nebo řetězec:

```
#define  MAX_ATOMS  1000
#define  PROGRAM_NAME  "Muj programek"
#define  SEPARATOR   '='
#define  NUMBER_PI   3.14159
#define  DOUBLE_PI   (2 * 3.14159)
```

- Jména symbolických konstant zpravidla zapisujeme velkými písmeny
- Symbolické konstanty slouží ke zpřehlednění k programu, navíc umožňují snadnou změnu jejich hodnoty na jednom místě (v definici symbolické konstanty)
- Pokud je to možné, dáváme v praxi přednost použití globálních proměnných místo symbolických konstant (většinou je pak definujeme jako konstantní pomocí `const`), např:

```
const char[] program_name =  "Muj programek";
```

- Symbolické konstanty používáme tam, kde nelze či není vhodné proměnnou použít (např. při specifikaci velikosti pole, v příkazu `switch`), nebo by použití proměnné vedlo ke zpomalení programu

Direktivy podmíněného překladu

- Direktiva #if zajistí, že určitá část programu bude překládána jen při splnění podmínky (jinak se ignoruje)
- Direktiva #if se často kombinuje s výrazem defined(JMENO_MAKRA), který vrací pravdivou hodnotu, pokud bylo makro JMENO_MAKRA dříve definováno pomocí #define; existuje též zkrácená forma #ifdef a #ifndef
- Direktiva #error zajistí výpis textové zprávy a přerušení překladu:
  #error Tady je nejaky text, ktery kompilator vypise na vystup

```
#if 0

// Zde muze byt nejaky kod v jazyce C, ktery chceme vynechat
// Zmenime-li 0 na 1, pak se kod opet bude prekladat

#endif
```

```
#define WIN32

#ifdef WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
```

Knihovny

- Knihovny jsou části programového kódu (především funkce) vytvořené jinými programátory pro použití v dalších programech
- Knihovna je tvořena souborem (pod UNIXem **.a*, **.so*, pod Windows **.lib*, **.dll*), soubory knihoven bývají zpravidla uloženy ve specifickém adresáři (v UNIXu především */usr/lib*, pod Windows především *C:\windows\system32*)
- Pod UNIXem začíná název souboru knihovny vždy slovem *lib*, pak následuje jméno knihovny, koncovka (*.a* nebo .so) a případně specifikace verze (např.: *libc.a*, *libm.so*, *libGL.so.1.0*)
- Ke knihovnám jsou dodávány hlavičkové soubory (*.h) které obsahují deklarace funkcí obsažených v knihovně. K jedné knihovně může příslušet několik hlavičkových souborů, hlavičkové soubory knihoven se pod UNIXem nachází především v adresáři */usr/include* a jeho podadresářích
- Při použití knihovny musíme vložit hlavičkové soubory pomocí direktivy #include a při překladu specifikovat jméno knihovny
- Více informací: http://en.wikipedia.org/wiki/Library_(computing)

Standardní knihovna jazyka C

- Součást všech překladačů jazyka C je standardní knihovna jazyka C, která obsahuje funkce pro práci se soubory, řetězci a podobně
- Standardní knihovna se automaticky připojí ke každému programu, není potřeba ji specifikovat, musíme však vždy na začátek programu vložit příslušný hlavičkový soubor, např.:

  assert.h - obsahuje makro assert() pro ladění programu
  ctype.h - práce se znaky
  errno.h - číselné kódy chyb
  float.h - makra definující max. a min. pro reálná čísla a pod.
  limits.h - makra definující max. a min. pro celá čísla a pod.
  stdio.h - souborový a standardní vstup a výstup
  stdlib.h - základní knihovní funkce, alokace paměti a pod.
  string.h - funkce pro práci s řetězci
  time.h - funkce pro práci s časem

- Více informací: https://en.wikipedia.org/wiki/C_library
- K překladačům jazyka C je také standardně dodávána matematická knihovna, hlavičkový soubor pro matematickou knihovnu je math.h
- Pro použití funkcí matematické knihovny musíme vložit hlavičkový soubor a při kompilaci použít parametr `-lm` (malé L)
- Seznam matematických funkcí: https://en.wikipedia.org/wiki/C_mathematical_functions

Make

- Nástroj *make* slouží k automatizovanému překladu programů
- Překlad se řídí instrukcemi zapsanými v souboru *Makefile* nebo *makefile*
- Jméno cíle (tj. jméno spustitelného souboru) zakončujeme dvojtečkou, pak uvádíme jména závislostí (prerekvizit) oddělených mezerou - jedná se zpravidla o zdrojový soubor s programem
- Dále uvádíme příkazy, každý na samostatném řádku, na začátku řádku musí být znak tabulátoru
- Překlad spustíme příkazem *make jmeno_cile*, spustíme-li *make* bez uvedení názvu cíle, provede se první cíl
- Více informací:
  https://en.wikipedia.org/wiki/Make_(software)

```
# Soubor muze obsahovat komentare, ktere zacinaji krizkem
# Budeme kompilovat soubor helloworld.c a vytvorime
# spustitelny soubor helloworld

helloworld: helloworld.c
    gcc -o helloworld helloworld.c
```

Zde musí být tabulátor, ne mezery!

Parametry překladače gcc

- Pro překlad programu v jazyce C používáme pod UNIXem zpravidla překladač *gcc*: `gcc -o spustitelny_soubor zdrojovy_soubor.c`
- Při překladu lze navíc použít následující parametry:

  `-Wall` – vypisuje upozornění (warnings) na možné problémy v programu

  `-pedantic` – vypisuje upozornění požadovaná standardem ISO C

  `-std=`*standard* – specifikuje standard jazyka C, ve kterém je program napsán (např.: -std=c89, -std=c99, -std=c11)

  `-I`*dir* – (velké I) přidá adresář *dir* do seznamu adresářů, ve kterých se hledají standardní hlavičkové soubory (tj. soubory vložené pomocí `#include <file.h>`, nikoliv `#include "file.h"`)

  `-L`*dir* – přidá adresář *dir* do seznamu adresářů, ve kterých se hledají knihovny

  `-l`*name* – (malé L) přidá knihovnu, knihovna se musí nacházet v systémových adresářích nebo adresáři specifikovaném -L, soubor s knihovnou se musí jmenovat lib`name`.a či lib`name`.so
- Při vývoji programů je vhodné používat parametry -std=c99 -Wall a -pedantic které upozorní na problémy v programu
- Další užitečné parametry jsou -o, -c, -D, -g, -s, -O (více: *man gcc*)

Knihovna g2

- Knihovna g2 nabízí funkce pro 2D kreslení a tak umožňuje tvorbu jednoduchých grafických programů
- Při kompilaci musíme specifikovat použití knihoven pomocí parametru `-l` (malé L). Kromě knihovny **g2** můžeme v našem kódu potřebovat například matematickou knihovnu, tedy: `-lg2 -lm`
- Celý příkazový řádek může vypadat např. následovně:

```
gcc -std=c99 -Wall -pedantic -o spustitelny_soubor
    zdrojovy_soubor.c -lg2 -lm
```

Knihovna g2

- Pro základní práci s knihovnou g2 v UNIXovém prostředí je potřeba vložit hlavičkové soubory *g2.h* a *g2_X11.h*
- Funkcí g2_open_X11() otevřeme okno, do něhož budeme kreslit; funkce přijímá dva parametry (šířku a výšku okna v pixelech), vrací číslo identifikující výstupní zařízení (tj. okno)
  int g2_open_X11(int width, int height)
- Okno zavřeme funkcí g2_close()
  void g2_close(int win)
- Dokumentace: http://www.ncbr.muni.cz/~martinp/g2/modules.html

```
#include <g2.h>
#include <g2_X11.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  int win = 0;
  win = g2_open_X11(400, 300);        // Otevreme okno
  g2_line(win, 10, 10, 390, 290);     // Kreslime caru
  getchar();          // Cekame na vstup znaku z klavesnice
  g2_close(win);      // Zavreme okno
  return 0;
}
```

Knihovna g2 – funkce pro kreslení

void g2_line(int win, double x1, double y1, double x2, double y2)

- Kreslí čáru z bodu x1, y1 do bodu x2, y2
- Parametr `win` je identifikátor okna (tj. návratová hodnota funkce `g2_open_X11()`)
- Parametry x1, y1, x2, y2 jsou souřadnice v pixelech (obrazových bodech), při kreslení do okna jsou vždy celočíselné (i když jsou typu `double`), levý dolní roh okna má souřadnice 0, 0

void g2_set_line_width(int win, double w)

- Nastaví šířku čáry v pixelech, kterou budou používat kreslící funkce

void g2_rectangle(int win, double x1, double y1,
double x2, double y2)

- Kreslí obdélník s protilehlými rohy v pozicích x1, y1 a x2, y2

void g2_filled_rectangle(int win, double x1, double y1,
double x2, double y2)

- Kreslí vyplněný obdélník

Knihovna g2 – funkce pro kreslení

```
void g2_circle(int win, double x, double y, double r)
void g2_filled_circle(int win, double x, double y, double r)
```

- Kreslí kružnici se středem v x, y a poloměrem r

```
void g2_ellipse(int win, double x, double y, double r1, double r2)
void g2_filled_ellipse(int win, double x, double y,
                       double r1, double r2)
```

- Kreslí elipsu se středem v x, y a délkou hlavní a vedlejší poloosy r1 a r2

```
void g2_arc(int win, double x, double y,
                       double r1, double r2, double a1, double, a2)
void g2_filled_arc(int win, double x, double y,
                       double r1, double r2, double a1, double, a2)
```

- Kreslí eliptickou výseč se středem elipsy v x, y, délkou hlavní a vedlejší poloosy r1 a r2, počátečním úhlem a1 a koncovým úhlem a2 (úhly se udávají ve stupních, tj. v rozsahu 0 až 360)

- Funkce obsahující v názvu `filled` kreslí vždy vyplněný obrazec

Knihovna g2 – funkce pro kreslení

void g2_set_font_size(int win, double size)

- Nastaví velikost fontu pro kreslení textu (velikost je v pixelech)

void g2_string(int win, double x, double y, const char *text)

- Kreslí text řetězce *text* na pozici x,y

void g2_pen(int win, int color)

- Nastaví aktuální barvu kterou pro kreslení, kterou použijí všechny dále volané kreslící funkce; hodnoty 0 až 26 reprezentují předdefinované barvy (např. 0 bílá, 1 černá, 3 modrá, 7 zelená, 19 červená, 25 žlutá), dále lze definovat uživatelské barvy pomocí `g2_ink()`

int g2_ink(int win, double red, double green, double blue)

- Vytvoří uživatelsky definovanou barvu na základě hodnot intenzity červené, zelené a modré (hodnoty v rozsahu 0.0 až 1.0), vrací číslo identifikující barvu, které lze použít ve funkci `g2_pen()`

Knihovna g2 – praktická práce s barvami

- Pokud používáme uživatelsky definované barvy, musíme nejdříve barvu vytvořit funkcí `g2_ink()` a potom ji nastavit funkcí `g2_pen()`, pak kreslíme objekty příslušnou barvou
- Můžeme použít dva přístupy:
  - pro každou barvu definujeme proměnnou, uložíme do ní číslo barvy vytvořené funkcí `g2_ink()` a potom tyto proměnné používáme při volání `g2_pen()`:

```
int color1 = 0, color2 = 0;
color1 = g2_ink(win, 0.5, 1.0, 1.0);
color2 = g2_ink(win, 1.0, 0.4, 0.8);
g2_pen(win, color1);
// Nasleduji kreslici funkce, kresli se barvou color1
g2_pen(win, color2);
// Kresleni barvou color2
```

  - případně můžeme `g2_ink()` volat přímo v argumentu `g2_pen()`:

```
g2_pen(win, g2_ink(win, 0.5, 1.0, 1.0));
// Nasleduji kreslici funkce, ktere budou
// kreslit prislusnou barvou
```

Dodržujte následující pravidla

- Pro překlad programů používejte nástroj *make*
- Vyplněné objekty vykreslujte s tloušťkou čáry 0, jinak by byl objekt větší o polovinu tloušťky čáry
- Při použití operátoru dělení si ujasněte, zda dochází k celočíselnému nebo reálnému dělení

Úlohy – část 1

1. Vytvořte program, který pomocí knihovny g2 nakreslí vše přibližně tak, jak je uvedeno na prvním obrázku. **3 body**
2. Vytvořte program, který nakreslí kruh se střídajícími se modrými a červenými segmenty (viz druhý obrázek). Počet segmentů bude specifikován na příkazovém řádku. **1 bod**
3. Vytvořte program, který nakreslí segmenty se střídající se modrou a červenou barvou (viz třetí obrázek). Celkový počet segmentů a počet segmentů na jednom řádku bude specifikován na příkazovém řádku (obrázek dole je tvořen 220 segmenty, 30 na řádku). **1 bod**