2. Základy programování v C

B0B99PRPA – Procedurální programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přehled témat

- Část 1 – Programování v jazyce C
  - Proměnné
  - Základní datové typy
  - Výrazy a operátory
  - Formátovaný vstup a výstup
- Část 2 – Funkce
- Část 3 – Překlad a sestavení programu
  - Překlad
  - Chyby při překladu

Část I

Programování v jazyce C

Me:

I am good in C language.

Interviewer:

Then write "Hello World" using C.

Me:

I. Programování v jazyce C

Příklad součtu dvou hodnot

lec02/sum1.c

```c
#include <stdio.h>

int main(void)
{
  /* definice lokalni promenne typu int */
  int sum;
  /* hodnota vyrazu se ulozi do sum */
  sum = 100 + 43;
  printf("Soucet 100 a 43 je %i\n", sum);
  /* %i formatovaci prikaz pro tisk celeho cisla */
  return 0;
}
```

- Proměnná typu `int` reprezentuje celé číslo, jehož hodnota je uložena v paměti
- Jméno proměnné `sum` symbolizuje paměťové místo v rámci programu

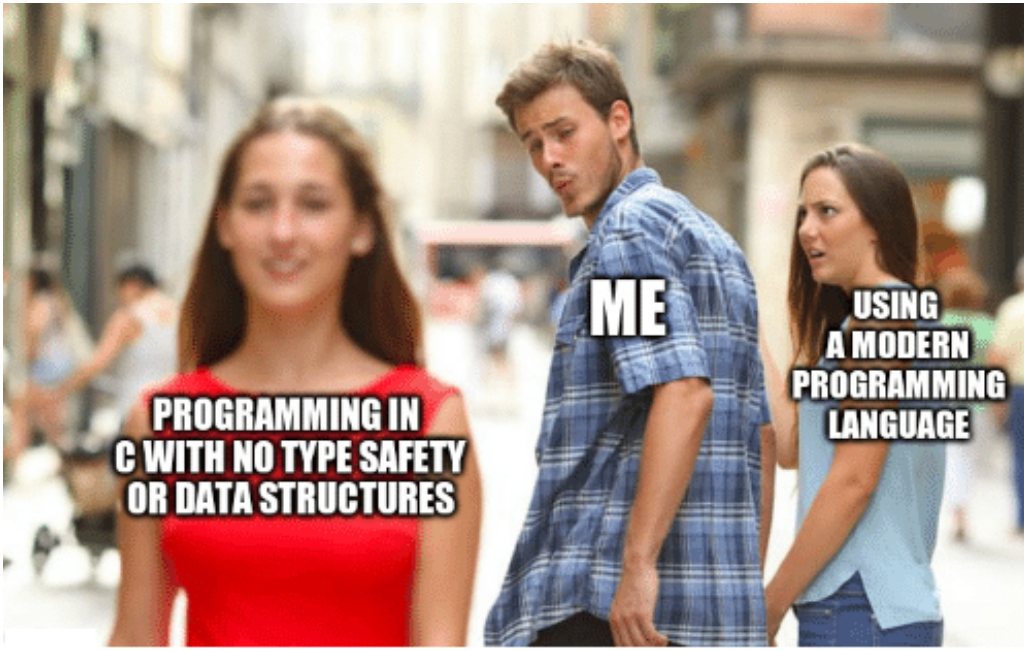
Příklad součtu hodnot dvou proměnných

lec02/sum2.c

```c
#include <stdio.h>

int main(void)
{
  int var1;
  int var2 = 10; /* inicializace hodnoty promenne */
  int sum;

  var1 = 13;
  sum = var1 + var2;

  printf("The sum of %i and %i is %i\n", var1, var2, sum);
  return 0;
}
```

- Proměnné `var1`, `var2` a `sum` reprezentují tři různá místa v paměti (automaticky přidělené), ve kterých jsou uloženy tři celočíselné hodnoty.

I. Programování v jazyce C


PROGRAMMING IN
C WITH NO TYPE SAFETY
OR DATA STRUCTURES
ME
USING
A MODERN
PROGRAMMING
LANGUAGE

Datové typy

Datový typ objektu (proměnná, pole...) určuje:

- Hodnoty, kterých může objekt nabývat
  - `float` může obsahovat reálná čísla
- Operace které lze (resp. nelze) nad objektem provádět
  - k celému číslu lze přičíst `10`
  - pole nelze podělit dvěma

C má následující datové typy:

- Primitivní
  - numerické (`int`, `float`, `double` a jejich modifikátory)
  - znakový (`char`)
  - ukazatelový
  - logický (od C99)
- Strukturované
  - pole
  - definované uživatelem (`struct`, `union`, `enum` a operátor `typedef`)

Celočíselné datové typy

int **short**, **long**

lze zařadit i **char** – celé číslo v rozsahu jednoho bajtu nebo také znak

- velikost paměti alokované příslušnou (celo)číselnou proměnnou se může lišit dle architektury počítače nebo překladače
  - typ `int` má zpravidla velikost 4 bajty a to i na 64bitových systémech
  - typ `char` má vždy velikost 1 bajt
- aktuální velikost paměťové reprezentace lze zjistit operátorem `sizeof()`
  - argumentem operátoru je jméno typu nebo proměnné

```
1  int i;
2  printf("%lu\n", sizeof(int));
3  printf("ui size: %lu\n", sizeof(i));
```

Hodnoty celočíselných datových typů

Pokud není uvedena u celého čísla přípona, jde o hodnotu typu `int`.

- dekadický `123`, `450932`
- šestnáctkový (hexadecimální) `0x12`, `0xFAFF` (začíná `0x` nebo `0X`)
- osmickový (oktalový)) `0123`, `0567` (začíná `0`)
- unsigned `12345U` (přípona `U` nebo `u`)
- long `12345L` (přípona `L` nebo `l`)
- unsigned long `12345ul` (přípona `UL` nebo `ul`)
- binární `0b00110011` (začíná `0b` nebo `0B`)

Není součástí standardu, jedná se o rozšíření (implementované jak v gcc, tak v clang)

Neceločíselné datové typy

float, double

- Umožňují zobrazit aproximace racionálních čísel v určitém rozsahu a s určitou přesností
- Jsou dané implementací, většinou dle standardu IEEE-754-1985

  `float` 32-bit IEEE 754, single-precision floating number

  `double` 64-bit IEEE 754, double-precision floating number
- Reprezentace reálných čísel $n = m \times b^e$
- IEEE reprezentace 32bitového (4B) čísla `float` má
  - **znaménkový bit** – znaménko mantisy
  - **exponent** – 8 bitů, číslo v rozsahu $\langle -126; 127 \rangle$
    - není uložen se znaménkem, nýbrž posunut o hodnotu 127
    - takže exponent -126 je uložen jako hodnota 1, exponent 127 se stane 254 atd.
  - **mantisa** – 23 bitů, normalizována do intervalu $\langle 1.0; 2.0 \rangle$.

Více o vnitřní implementaci datových typů na 8. přednášce.

Hodnoty neceločíselných datových typů

Racionální číslo bez přípony je hodnotou typu `double`.

- `float` – prípona `F` nebo `f`
- `long double` – přípona `L` nebo `l`

Formát zápisu racionálních literálu

- s řádovou tečkou – `13.1`
- mantisa a exponent – `31.4e-3` nebo `31.4E-3`

Znakový datový typ

char

- Reprezentuje celé číslo (byte) nebo znak (gr. symbol)
- Kódování znaků ASCII – American Standard Code for Information Interchange.
  - znaménkový, implicitní specifikátor záleží na kompilátoru
  - lze s nimi používat aritmetické nebo relační operátory a pod.
  - hodnotu znaku lze zapsat jako tzv. znakovou konstantu

lec02/char.c

```
1  char C = 'a';
2  printf("Hodnota C=%i nebo znak '%c'\n", C, C);
3  C = C + 1;
4  printf("Hodnota C=%i nebo znak '%c'\n", C, C);
```

- Řídící znaky
  - \n – new line, \t – tabular, \r – carriage return
  - \a – beep, \b – backspace, \f – form feed, \v – vertical space

Hodnoty znakového datového typu

- Typ – znaková konstanta je typu `int`
- Formát – znak v jednoduchých apostrofech: `'A'`, `'B'` nebo `'\n'`
- Hodnota – znakový literál má hodnotu odpovídající kódu znaku: `'0'` ~ 48, `'A'` ~ 65

Hodnota znaku mimo ASCII (větší než 127) závisí na překladači.

- Specifikace číselné hodnoty v tabulce znaků (ASCII)
  - hodnota v oct nebo hex soustavě, uvozena apostrofy, před vlastní hodnotu zpětné lomítko
  - pokud neuvedeme na začátku 0, je hodnota interpretována jako číslo v osmičkové soustavě tj. `'\32'` je totéž jako `'\032'`, tedy hodnota 32 v osmičkové soustavě (26 v desítkové)

  Příklad řídící znaky. `'\110'`, `'\07'`, `'\0xA3'`
- Zajímavosti v ASCII tabulce

dec	bin	hex	ascii
32	00100000	0x20	`' '`
48	00110000	0x30	`'0'`
57	00111001	0x39	`'9'`
65	01000001	0x41	`'A'`
97	01100001	0x61	`'a'`

Změnou bitu lze změnit velikost znaku [A-Za-z]

Odečtením konstanty lze získat číselnou hodnotu

Logický datový typ

- Ve verzi **C99** je zaveden logický datový typ `_Bool`
- Jako hodnota true je libovolná hodnota typu `int` různá od 0

Tento přístup je zcela dostatečný.

- Lze využít hlavičkového souboru `<stdool.h>`, kde je definován typ `bool` a hodnoty true a false

```
#define false 0
#define true 1
#define bool _Bool
```

- **ANSI C** explicitní datový typ pro logickou hodnotu nedefinuje.
- Běžně se používá podobná definice jako v `<stdbool.h>`

```
#define FALSE 0
#define TRUE 1
```

Modifikátory délky numerických datových typů

short, long, long long

- Modifikátory mění rozsah oběma směry, nemusí být nutně možné všechny kombinace
- **short** — kratší verze typu
  - **short int** ≡ **short**
  - neceločíselné datové typy zkrácení neumožňují, od IEEE 754-2008 lze používat half-precision floating-point formát, využívá se např. v HDR pro úsporu místa
- **long** — delší verze typu (podpora překladače)
  - `long int` ≡ `long`
  - `long double` – 80bitová verze `double` (ve skutečnosti zabírá 96b), na některých platformách lze pracovat i se čtyřnásobnou přesností , kde je číslo reprezentováno 128 bity; často se ale typ jmenuje jinak – např. `double double` (SPARC) nebo `__float128` (gcc pro x86, x86-64 a Itanium)
- **long long** — ještě delší verze typu (podpora překladače)
  - od standardu C99
  - `unsigned long long` může reprezentovat číslo +18.446.744.073.709.551.615, na některých platformách případně i více

Rozsahy celočíselných typů

- Rozsahy celočíselných typů v C nejsou dány normou, ale implementací
  - Mohou se lišit implementací a prostředím 16 bitů vs. 32 bitů vs. 64 bitů
  - Bývá zvykem, že int je 4 bytový

Embedded platformy (8051, Arduino) mívají int 16 bitový.

- Norma garantuje, že pro rozsahy typů platí

$$\texttt{short} \leq \texttt{int} \leq \texttt{long}$$
$$\texttt{unsigned short} \leq \texttt{unsigned} \leq \texttt{unsigned long}$$

- Typy se specifickou velikostí jsou definované např. v `<stdint.h>`

IEEE Std 1003.1-2001, známá jako POSIX.1-2001

`int8_t`	`int16_t`	`uint32_t`
`uint8_t`	`uint16_t`	`uuint32_t`

I. Programování v jazyce C

Výraz

- Výraz předepisuje výpočet hodnoty určitého vstupu
- Struktura výrazu obsahuje operandy, operátory a závorky
- Výraz může obsahovat
  - proměnné
  - operátory
  - volání funkcí
  - konstanty
  - závorky
- Pořadí operací předepsaných výrazem je dáno prioritou a asociativitou operátorů

Operátory

- Znaky (nebo posloupnost znaků) vyhrazené pro zápis výrazů
- Binární operátory

  Aritmetické sčítání, odčítání, násobení, dělení

  Relační porovnání hodnot (menší, větší, ...)

  Logické logický součet a součin

  Přiřazení na levé straně operátoru `=` je proměnná
- Unární operátory
  - indikující kladnou/zápornou hodnotu: `+` a `-` (operátor `-` modifikuje znaménko výrazu za ním)
  - modifikující promenou: `++` a `--`
  - logický operátor doplněk: `!`
  - operátor pretypování: `(jméno typu)`
- Ternární operátor
  - podmíněné přiřazení hodnoty

Proměnné a přiřazení

- Proměnné deklarujeme uvedením typu a jména proměnné
  - Jména proměnných volíme malá písmena
  - Víceslovná jména zapisujeme s podtržítkem (nebo **CamelCase**)
  - Proměnné definujeme na samostatném řádku

```
1  int n;
2  int number_of_items;
3  int LongNameOfVariable;
```

- Přiřazení je nastavení hodnoty proměnné, tj. uložení hodnoty na místo v paměti, kterou proměnná reprezentuje
- Tvar přiřazovacího operátoru: **proměnná** = **výraz**

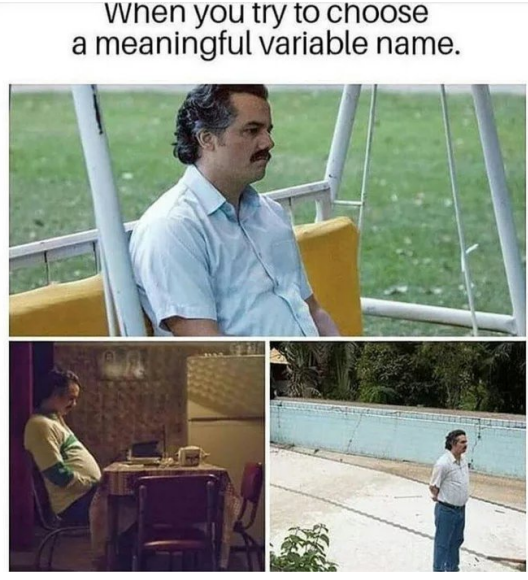
Výraz je literál, proměnná, volání funkce,...

- **Příkaz přiřazení** se skládá z operátoru přiřazení = a ;
  - Levá strana přiřazení musí být **l-value** – location-value, left-value

Tj. musí reprezentovat paměťové místo pro uložení výsledku.

  - Přiřazení je výraz a můžeme jej použít všude, kde je dovolen výraz příslušného typu

Názvy proměnných



P2.1 Operátor přiřazení

lec02/swap-tmp.c

```c
#include <stdio.h>

int main()
{
  int a = 3, b = 7;
  int tmp;
  tmp = a;
  a = b;
  b = tmp;
  return 0;
}
```

lec02/swap.c

```c
#include <stdio.h>

int main()
{
  int a = 3, b = 7;
  a = a + b;
  b = a - b;
  a = a - b;
  return 0;
}
```

Aritmetické operátory

- Pro operandy číselných typů jsou definovány operátory
  - unární operátor změna znaménka -
  - binární sčítání +, odčítání −, násobení ∗ a dělení /
- Pro operandy celočíselných typů pak dále
  - binární operátor zbytek po dělení %
- Pro oba operandy stejného typu je výsledek aritmetické operace stejného typu
- V případě kombinace typu **int** a **double**, se **int** převede na **double** a výsledek je hodnota typu double.

Implicitní typová konverze.

P2.2 Aritmetické operátory

lec02/arith.c

```c
int a = 10;
int b = 3, c = 4;
int d = 5, result;

result = a - b; // rozdil
printf("a - b = %i\n", result);
result = a * b; // nasobeni
printf("a * b = %i\n", result);
result = a / b; // celociselne deleni
printf("a / b = %i\n", result);

result = a + b * c; // priorita operatoru
printf("a + b * c = %i\n", result);

printf("a * b + c * d = %i\n", a * b + c * d); // 50
printf("(a*b) + (c*d) = %i\n", (a*b) + (c*d)); // 50
printf("a * (b+c) * d = %i\n", a * (b+c) * d); // 350
```

P2.3 Aritmetické operátory

lec02/points.c

```c
int x1 = 1;
double y1 = 2.2357;
float x2 = 2.5343f;
double y2 = 2;

printf("P1 = (%i, %f)\n", x1, y1);
printf("P1 = (%i, %i)\n", x1, (int)y1);
// operator pretypovani (double)
printf("P1 = (%f, %f)\n", (double)x1, (double)y1);
printf("P1 = (%.3f, %.3f)\n", (double)x1, (double)y1);

printf("P2 = (%f, %f)\n", x2, y2);
// implicitni konverze na float, resp.  double
double dx = (x1 - x2);
double dy = (y1 - y2);

printf("(P1 - P2) = (%.3f, %0.3f)\n", dx, dy);
printf("|P1 - P2|^2 = %.2f\n", dx * dx + dy * dy);
```

I. Programování v jazyce C

Standardní vstup a výstup

- Program spuštěný v prostředí operačního systému má přístup ke znakově orientovanému standardnímu vstupu (`stdin`) a výstupu (`stdout`)

  Jsou to prostředky operačního systému.

- Prostřednictvím `stdout` a `stdin` program komunikuje s uživatelem

  U MCU probíhá komunikace jinými prostředky (např. sériový port).

- Funkce definované v knihovně `<stdio.h>`
  - `putchar()` – výstup (tisk) znaku
  - `getchar()` – vstup (načtení) znaku
  - `printf()` – výstup formátovaného textu
  - `scanf()` – vstup formátovaného textu

Standardní vstup a výstup

- Program spuštěný v prostředí operačního systému má přístup ke znakově orientovanému standardnímu vstupu (`stdin`) a výstupu (`stdout`)

  Jsou to prostředky operačního systému.

- Prostřednictvím `stdout` a `stdin` program komunikuje s uživatelem

  U MCU probíhá komunikace jinými prostředky (např. sériový port).

- Funkce definované v knihovně `<stdio.h>`
  - `putchar()` – výstup (tisk) znaku
  - `getchar()` – vstup (načtení) znaku
  - `printf()` – výstup formátovaného textu
  - `scanf()` – vstup formátovaného textu

Výstup znaku putchar

- Vytiskne na standardní výstup jeden znak
  - hodnota datového typu `char`
  - znak odpovídající hodnotě celočíselného typu v ASCII tabulce

Příklad

```c
#include <stdio.h>

int main()
{
  char a = 'A';
  int b = 8;

  putchar(a);
  putchar(a++);
  putchar(b);
  putchar(b+80);
}
```

```
AX
```

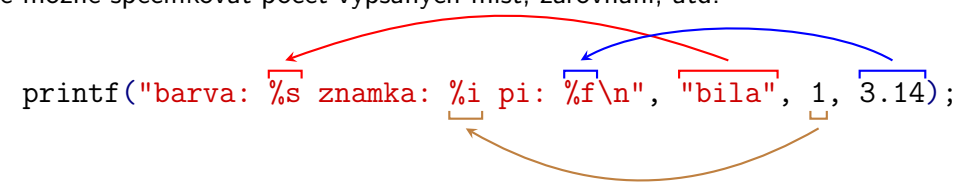
lec02/putchar.c

Formátovaný výstup printf

- Na rozdíl od `putc()` umí tisknout obsah proměnných různých datových typů
- Argumentem funkce je textový řídicí řetězec formátování výstupu
  - řídicí řetězec formátu je uvozen znakem '%'
  - znakové posloupnosti (nezačínající %) se vypíší tak jak jsou uvedeny
  - důležité řídicí řetězce pro výpis hodnot proměnných:

char	%c
_Bool	%i, %u
int	%i, %x, %o
float	%f, %e, %g, %a
double	%f, %e, %g, %a

- Je možné specifikovat počet vypsaných míst, zarovnání, atd.



P2.4 Formátovaný výstup

lec02/printf.c

```c
#include <stdio.h>

int main(void)
{
  int i = 'a';
  int h = 0x61; /* sestnactkova soustava */
  int o = 0141; /* osmickova soustava */
  printf("i: %i h: %i o: %i c: %c\n", i, h, o, i);
  printf("oct: \141 hex: \x61\n");
  return 0;
}
```

Vstup znaku getchar

- Přečte jeden znak ze standarního vstupu
- HW rozhraním mezi standardním vstupem a programem je buffer
  - Co se stane, pokud uživatel zadá více než jeden znak?
  - Co se stane, pokud uživatel nezadá žádný znak (jen stiskne Enter)?

lec02/getchar.c

```
1  #include <stdio.h>
3  int main () {
4    char c;
6    printf("Zadejte znak: ");
7    c = getchar();
9    printf("Zadany znak: %c %i\n", c, c);
11   return 0;
12 }
```

Formátovaný vstup scanf

- Načtení obecně textové hodnoty ze standardního vstupu a konverze na vhodný datový typ
- Argumentem je textový řídicí řetězec, podobně jako u funkce `printf()`
- Je nutné předat adresu paměťového místa pro uložení hodnoty

lec02/scanf.c

```
1  #include <stdio.h>
3  int main(void)
4  {
5    int i;
7    // program ceka na data, operator & vraci adresu promenne i
8    // na kterou se ulozi nactena hodnota
9    scanf("%i", &i);
10   printf("Zadal jsi %in", i);
12   return 0;
13 }
```

Část II

Funkce a modularita

Funkce

- Funkce tvoří základní stavební blok **modulárního** jazyka C

  Modulární program je složen z více modulů/zdrojových souborů.

- Každý spustitelný program v C obsahuje alespon jednu funkci a to funkci `main()`
- **Deklarace** – hlavička funkce (prototyp)

```
navratovy_typ jmeno_funkce (formalni parametry);
```

  Deklarace obsahuje informace, které vyžaduje překladač.

- **Definice** – hlavička funkce a její tělo (sekvence příkazů)

```
navratovy_typ jmeno_funkce (formalni parametry) {
  // tělo funkce
  return hodnota_navratoveho_typu;
}
```

  Definice funkce bez předchozí deklarace je zároveň deklarací funkce.

Vlastnosti funkcí

- C nepovoluje funkce vnořené do jiných funkcí
- Jména funkcí se mohou exportovat do ostatních modulů (souborů)
- Formální parametry funkce jsou lokální proměnné, které jsou inicializovány skutečnými parametry při volání funkce

Parametry se do funkce předávají hodnotou

- C dovoluje **rekurzi** – funkce může volat sebe samu
- Funkce nemusí mít žádné vstupní parametry, zapisujeme:

```
navratovy_typ jmeno_funkce (void) {
  // tělo funkce
  return hodnota_navratoveho_typu;
}
```

- Funkce nemusí vracet funkční hodnotu – návratový typ je pak `void`

I v takovém případě může obsahovat **return**, ovšem bez argumentu

2.5 Struktura modulu

lec03/modul.c

```c
#include <stdio.h>  /* hlavickovy soubor */

#define PI 3.14     /* symbolicka konstanta - makro */

float kruh(int a);  /* hlavicka/prototyp funkce */

int main(void)      /* hlavni funkce */
{
  int v = 10;       /* definice promennych */
  float r;
  r = kruh(v);      /* volani funkce */
  return 0;         /* ukonceni hlavni funkce */
}

float kruh(int r)   /* definice funkce */
{
  float b = PI*r*r;
  return b;        /* navratova hodnota funkce */
}
```

Zdrojové a hlavičkové soubory

- Rozsáhlejší programy je vhodné rozdělit do více souborů – **modulů**
- Rozdělení na zdrojové a hlavičkové soubory umožňuje rozlišit **definici** a **deklaraci**
- Dělením do modulů je podporována
  - **Organizace** zdrojových kódů v adresářové struktuře souborů
  - **Modularita**
    - **Hlavičkový soubor** – obsahuje popis (seznam) funkcí a jejich parametrů bez konkrétní implementace (deklarace funkcí)

      Bývá zvykem do hlavičkových souborů umisťovat i definice datových typů, konstanty a makra.
    - **Zdrojový soubor** – konkrétní implementace funkcí modulu

      Optimalizace překladu – pokud se modul od posledního překladu nezměnil, není třeba zdrojový kód modulu znova překládat
  - **Znovupoužitelnost**
    - Pro využití **binární knihovny** je třeba znát její **rozhraní** deklarované v hlavičkovém souboru

      Výhodné pro práci na velkých projektech nebo týmech

Část III

Překlad a sestavení programu

III. Překlad a sestavení programu

Překlad

Chyby při překladu

Překlad a sestavení programu

- Překlad je posloupností tří dílčích úloh, které se zpravidla provádí automaticky
- Jednotlivé kroky překladu lze provést i individuálně

1. Textové předzpracování **preprocesorem** (má vlastní makro jazyk, příkazy uvozeny znakem #)

   Odkazované hlavičkové soubory se vloží do jediného zdrojového souboru

   ```
   $ gcc -E program.c -o program.i
   ```

2. Vlastní **překlad** zdrojového souboru do objektového souboru

   Zpravidla jsou jména souboru zakoncena príponou .o

   ```
   $ gcc -c program.c -o program.o
   ```

   Příkaz kombinuje volání preprocesoru a kompilátoru.

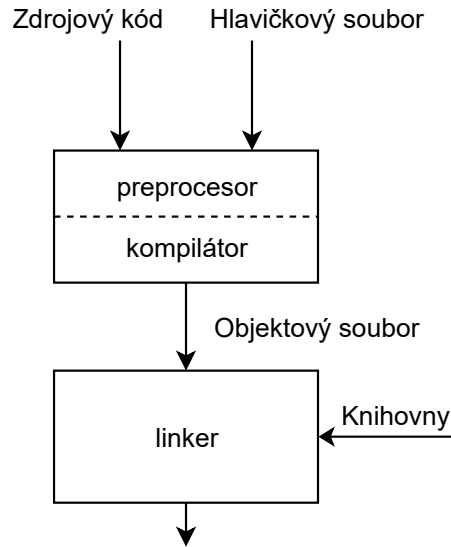
3. Spustitelný soubor se sestaví z příslušných dílčích objektových souborů a odkazovaných knihoven, tzv. linkováním (**linker**), např.

   ```
   $ gcc program.o -o program
   ```

Schéma překladu a sestavení programu

Vývoj programu v C

- editace zdrojových souborů
- kompilace dílčích zdrojových souborů do objektových souborů
- linkování přeložených souborů do spustitelného programu
- spouštění a ladění aplikace
- opětovná editace zdrojových souborů



Dílčí kroky překladu a sestavení programu

preprocesor umožňuje definovat makra a tím přizpůsobit překlad aplikace kompilačnímu prostředí

Výstupem je zdrojový soubor – text.

kompilátor překládá zdrojový (textový) soubor do strojově čitelné (a spustitelné) podoby

Nativní (strojový) kód platformy, objektový soubor má pouze relativní adresy volání funkcí.

linker sestavuje program z objektových souborů do podoby výsledné aplikace

Může odkazovat na knihovní funkce (dynamické knihovny linkované při spuštění programu) nebo volání OS (knihovny). Relativní adresy jsou nahrazeny absolutními.

Dílčí části (preprocesor, kompilátor a linker) jsou zpravidla součástí jednoho programu (clang, gcc), který lze spouštět s různými parametry.


RUN GCC
RUN
memegenerator.net

Překladače jazyka C

- V rámci předmětu budeme používat především překladače
  - **gcc** – GNU Compiler Collection

    https://gcc.gnu.org
  - **clang** – C language family frontend for LLVM

    http://clang.llvm.org

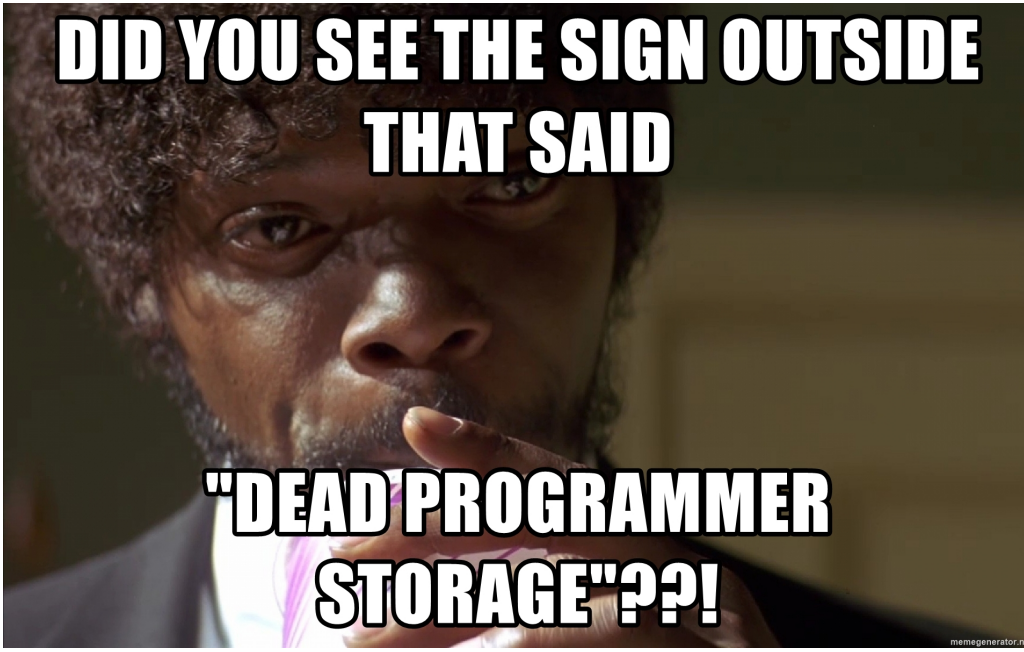
  Základní použití (přepínače a argumenty) je u obou překladačů stejné.
- Pro win* platformy existují odvozená prostředí cygwin nebo MinGW

  https://www.cygwin.com/, http://www.mingw.org/

Příklad

```
$ gcc -c program.c -o program.o
$ gcc program.o -o program
```

III. Překlad a sestavení programu


DID YOU SEE THE SIGN OUTSIDE THAT SAID
"DEAD PROGRAMMER STORAGE"??!
memegenerator.net

Chybová hlášení

- Analýze chybových hlášení je třeba věnovat zvýšenou pozornost
- Obvykle zjistíte přesný popis problému

A ještě se trochu pocvičíte v angličtině.

- Překládejte se zapnutým hlášením všech chyb

```
$ gcc -Wall ...
```

```
hw2.c : 87 : 7 :   error :   ‘foo’ undeclared
```

- Soubor
- Číslo řádky
- Pozice písmena
- Klasifikace (error nebo warning)
- Samotné chybové hlášení

Chybová hlášení

- Analýze chybových hlášení je třeba věnovat zvýšenou pozornost
- Obvykle zjistíte přesný popis problému

A ještě se trochu pocvičíte v angličtině.

- Překládejte se zapnutým hlášením všech chyb

```
$ gcc -Wall ...
```

```
hw2.c : 87 : 7 :   error :   ‘foo’ undeclared
```

- Soubor
- Číslo řádky
- Pozice písmena
- Klasifikace (error nebo warning)
- Samotné chybové hlášení

Chybová hlášení

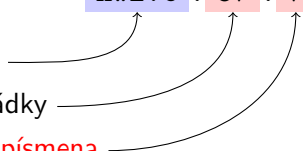
- Analýze chybových hlášení je třeba věnovat zvýšenou pozornost
- Obvykle zjistíte přesný popis problému

A ještě se trochu pocvičíte v angličtině.

- Překládejte se zapnutým hlášením všech chyb

```
$ gcc -Wall ...
```

```
hw2.c : 87 : 7 :   error :   ‘foo’ undeclared
```



- Soubor
- Číslo řádky
- Pozice písmena
- Klasifikace (error nebo warning)
- Samotné chybové hlášení

Chybová hlášení

- Analýze chybových hlášení je třeba věnovat zvýšenou pozornost
- Obvykle zjistíte přesný popis problému

A ještě se trochu pocvičíte v angličtině.

- Překládejte se zapnutým hlášením všech chyb

```
$ gcc -Wall ...
```

```
hw2.c : 87 : 7 :   error :   ‘foo’ undeclared
```

- Soubor
- Číslo řádky
- Pozice písmena
- Klasifikace (error nebo warning)
- Samotné chybové hlášení

Chybová hlášení

- Analýze chybových hlášení je třeba věnovat zvýšenou pozornost
- Obvykle zjistíte přesný popis problému

A ještě se trochu pocvičíte v angličtině.

- Překládejte se zapnutým hlášením všech chyb

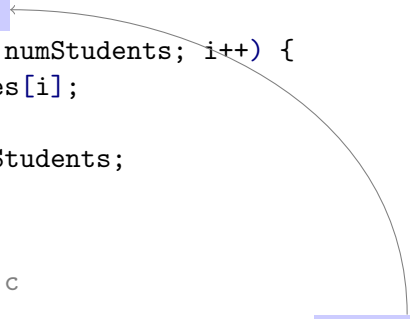
```
$ gcc -Wall ...
```

```
hw2.c : 87 : 7 :   error :   ‘foo’ undeclared
```

- Soubor
- Číslo řádky
- Pozice písmena
- Klasifikace (error nebo warning)
- Samotné chybové hlášení

Kaskádování chyb

```
1  int numStudnts;
2  for (i = 0; i < numStudents; i++) {
3    total += grades[i];
4  }
5  avg = total/numStudents;
```



```
$ gcc -Wall average.c
average.c:1:5: warning: unused variable ‘numStudnts’
average.c:2:17: error: ‘numStudents’ undeclared
average.c:5:13: error: ‘numStudents’ undeclared
```

Po opravě první chyby ve výpisu jsou ošetřeny i ostatní chyby.

Chyby při kompilaci

- většinou překlepy a jednodušší chyby

`hw2.c:37:6: warning: unused variable 'bar'`

  - proměnné je deklarována, ale nepoužita
  - typicky překlep, nebo nedokončená myšlenka

`hw2.c:54: warning: suggest parentheses around assignment used as truth value`

  - častá chyba při psaní podmínek
  - záměna operátorů `==` a `=`

`hw2.c: 51: error: expected ';' before 'for'`

  - chybějící znak `;` na **předchozím** řádku
  - místo `for` může být jiné klíčové slovo

Chyby při linkování

- mohou být trochu těžší na odhalení

```
hw4.o:  In function ‘main’:
hw4.c:91:  undefined reference to ‘Fxn’
```

  - linker nemůže najít kód funkce ‘Fxn’ v žádném objektovém souboru
    - není přilinkován správný objektový soubor
    - špatně zapsané jméno funkce Fxn
    - jiný než očekávaný seznam parametrů funkce
    - rozdíly mezi prototypem / definicí / voláním

```
/usr/lib64/gcc/[...]/crt1.o:  In function ‘_start’:
[...]/start.S:119:  undefined reference to main
```

  - zdrojový soubor neobsahuje funkci `main`
  - při separátní kompilaci modulu bez volby **-c**

Shrnutí přednášky

Diskutovaná témata

- Překlad programu v C
- Proměnné
- Základní datové typy
- Výrazy a operátory
- Vstup a výstup
- Funkce

- Příště: Základní řídicí struktury

Diskutovaná témata

- Překlad programu v C
- Proměnné
- Základní datové typy
- Výrazy a operátory
- Vstup a výstup
- Funkce

- Příště: Základní řídicí struktury