Programovací jazyk C(++)

```
static struct vm_area_struct * unmap_fixup(struct mm_struct *mm,
        struct vm_area_struct *area, unsigned long addr, size_t len,
        struct vm_area_struct *extra)
{
        struct vm_area_struct *mpnt;
        unsigned long end = addr + len;

        area->vm_mm->total_vm -= len >> PAGE_SHIFT;
        if (area->vm_flags & VM_LOCKED)
                area->vm_mm->locked_vm -= len >> PAGE_SHIFT;

        /* Unmapping the whole area. */
        if (addr == area->vm_start && end == area->vm_end) {
                if (area->vm_ops && area->vm_ops->close)
                        area->vm_ops->close(area);
                if (area->vm_file)
                        fput(area->vm_file);
                kmem_cache_free(vm_area_cachep, area);
                return extra;
        }
```

1. Základy

Miroslav Spousta, 2004

Historie

- Jazyk C (K&R)

  Brian W. Kernighan a Denis M. Ritchie: The C Programming Language (1978)

  úzce spjat s vývojem UNIXu (je v něm napsána i většina jádra)

- ANSI C (C89 nebo C90)

  z roku 1989, dnes běžný, definuje jazyk a knihovny, které jsou povinné

  také jako ISO standard (1990)

- C++

  objektové rozšíření, Bjarne Soustrup, Bell Labs (1983)

  rozšiřován: přidány výjimky, templates

  ANSI C++ (1997), ISO (1998)

Jazyk C

- jazyk nízké úrovně

  neobyčejně silný, neobyčejně nebezpečný

  má úsporné vyjadřování (žádné begin, end)

  univerzální (od jádra operačního systému po aplikace, od sálových počítačů po mikročipy)

  jen základní datové typy (číslo, znak, nikoli řetězec)

- přenositelný (norma ANSI C)
- základ programátora, mateřský jazyk UNIXu
- kód může být velmi efektivní

  srovnatelný s assemblerem, máme-li dobrý kompilátor

Jazyk C++

- objektové rozšíření jazyka C
  - struktury s asociovanými funkcemi (metodami)
  - dědičnost, zapouzdření, polymorfismus
- šablony (templates)
- přetěžování operátorů a funkcí
- výjimky a jejich ošetření
- drobné odlišnosti (něco má navíc C, něco C++)
- průnik neprázdný

Hello world!

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   printf("Hello world\n");
   return 0;
}
```

- funkce: main(), printf()

  mají argumenty, návratový typ

- hlavičkový soubor: stdio.h

  s popisem používaných funkcí

Struktura programu

- kód je rozdělený do funkcí

  např. funkce main()

- funkce se skládají hlavičky a bloků kódu

  bloky jsou ve složených závorkách **{ … }**

  každá funkce má svůj blok, v něm mohou být vloženy další

- v bloku jsou uvedeny deklarace, za nimi kód

  v C99 a C++ se deklarace mohou s kódem míchat

(Oddělený) překlad

- kód programu

  zdrojový text: **`soubor.c`** nebo **`soubor.cc`** či **`soubor.cpp`**

  hlavičkové soubory: **`soubor.h`** nebo **`soubor.hh`**

- kompilátor

  vytvoří ke každému zdrojovému souboru objektový soubor: **`soubor.o`**

- linker

  spojí všechny objektové soubory, přidá knihovny

  vytvoří spustitelný soubor (inicializační kód)

Identifikátory

- identifikátor

  pojmenování něčeho (objektu) v program

  libovolná posloupnost písmen (a znaku „_“) a čísel začínající písmenem (nebo „_“)

  rozlišuje se velikost písmen**!**

  C: identifikátory psát malými písmeny a s podtržítky, C++: první písmeno slova velké

  nepoužívat podobné identifikátory (system, system_)

  nepoužívat stejné identifikátory lišící se jen velikostí písmen (kostra, Kostra, KOSTRA)

- komentáře

  ```
  /* komentář */
  // komentář
  ```

Datové typy

- čísla celá:

  `short, int, long, long long` (C99)

  `unsigned short, unsigned, unsinged long`

- znaky

  `signed char, unsigned char, char` (jedno z předchozích)

  `wchar_t` (unicode)

- čísla reálná (komplexní jen C99)

  `float, double, long double`

- logické hodnoty

  `bool` (C99, C++), klasicky v C se to řeší pomocí typu **`int`**

Datové typy 2

- velikost typu char je 1 bajt, jinak je definováno pouze:

  `short <= int <= long <= long long`

  `float <= double <= long double`

- konstanty

  celé (desítkově) -10, 4, 45, (šestnáctkově) -0xA 0x4 0x44, (osmičkově) 043, 0, -016

  reálné: **`15., .84, 7e10`**

  znaky: **`'a', '%', '\n', '\t'`**

  řetězce: **`"nejaky retezec", "nejaky" "jiny retezec"`**

- typ C: **`#define PI 3.1415926`**

- typ C++: **`const double PI = 3.1415926`**

Definice, deklarace

- deklarace: udává typ proměnné a její jméno

```
extern int parse;
```

- definice: deklarace, která navíc *alokuje paměť* (pro danou proměnnou)

```
int test;
```

- lokální proměnné: platí v daném bloku
- globální proměnné: platí v celém programu
- proměnnou můžeme při definici inicializovat

```
int test = 0;
```

Příklad

```
int g;          /* globální proměnná */

int f(int x)
{
   int y;       /* lokální proměnná */
   y = x * x + g;
   return y;
}

int main(int argc, char *argv[])
{
   g = 2;
   printf("5 * 5 + 2 je %d!\n", f(5));
}
```

funkce, main()

- v C jsou jen funkce (vždy se něco vrací, aspoň prázdný typ **`void`**)
- parametry funkcí mohou být proměnné libovolného typu

  např.: **`int sum(int a, int b)`**
- jedna funkce se vždy jmenuje **`main()`**

  je to první funkce, která se vykoná po spuštění programu, v programu musí být

  vrací typ **`int`** (návratovou hodnotu pro OS)

  má dva parametry počet vstupních argumentů a pole argumentů
- funkce se ukončuje příkazem **`return`** (s návratovou hodnotou funkce)

  případně přechodem přes konec bloku (funkce typu **`void`**)
- funkce se mohou volat rekurzivně

Příkaz

- výraz ukončený středníkem

  výraz: `i = 4`

  příkaz: `i = 4;`

- prázdný příkaz: jen středník

  např. tam, kde nějaký příkaz být musí (tělo cyklu)

Operátory

- sčítání, odčítání: `+, -`
- unární inkrementace, dekrementace: `++, --`
- násobení, dělení, zbytek po dělení: `*, /, %`
- logické operátory (and a or): `&&, ||`
- bitové operátory (and, or, xor): `&, |, ^`
- bitový posun vlevo, vpravo: `<<, >>`
- porovnání: `<, <=, >, >=, ==, !=`
- přiřazení: `=, +=, -=, *=, /=, %=, &=, |=, ^=, >>=, <<=`

Operátory

- l-value: proměnná nebo něco, kam je možné přiřazovat (má adresu)

POZOR

- přiřazení je jen `=`, ne `:=`
- `x = 4` je výraz, vrací hodnotu 4, pozor na záměnu s porovnáním (`==`)

  např. v `if (x = 4) { ... }` je podmínka vždy splněná
- na prioritu operátorů, závorkovat
- nulová hodnota je nepravda, nenulová pravda
- *unární operátory neoddělujeme mezerou (i++)*
- *po čárce, středníku píšeme mezeru, před nimi ne!*

Podmínka

- `if (výraz) příkaz; else příkaz;`
- místo příkazu může být i blok
- možno i bez else větve
- žádné „then“!
- nenula ve výrazu zamená true,

  nula je false
- zkrácené vyhodnocení výrazu

  stop, víme-li, jak výraz dopadne

```
int f(int a, int b)
{
   int c;
   if (b) {
      c = a % b;
   } else {
      c = 0;
   }
   return c;
}
```

Cykly

- **`while (výraz) { … }`**

  klasický while cyklus

- **`do { … } while (výraz)`**

  opakuje se, dokud je podmínka splněna (pozor, jinak než v Pascalu!)

- **`for (v1; v2; v3) { … }`**

  v1: inicializace

  v2: podmínka, vyhodnocuje se před každým cyklem

  v3: vyhodnocuje se po každém cyklu

```
int i = 0;

while (i++ < 10) {
   printf(".");
}

do {
   printf(".");
while (i++ < 10);

for(i = 0; i < 10; i++) {
   printf(".");
}
```

Ještě k cyklům

- vyskočení z cyklu: **`break`**
- další iterace: **`continue`**
- *je-li tělo cyklu prázdné, píšeme středník na novou řádku*
- operátor čárky: **`j = (1, 3);`**

  použití: např. inicializace ve for-cyklu

```
double i = -10;

while (i++ < 10) {
    if (i == 0)
        continue;
    printf("%.5f\n", 1/i);
}

for (i = 1; i < 10; i <<= 1)
    ;
printf("Vysledek: %d\n", i);
```

Switch

- testuje hodnotu dané celočíselné proměnné, podle jejích hodnot provádí kód
- **`break`** ukončuje switch
- **`continue`** ho nijak neovlivňuje!
- default větev se provádí, pro hodnoty, které nejsou mezi vyjmenovanými
- neuvedeme-li break, propadáme do další větve

```
void f(int i)
{
   switch (i) {
      case 1:
         puts("jedna");
         break;
      case 2:
         puts("dve");
      default:
         puts("honza de");
   }
}
f(1); f(2); f(3);
```

Terminálový vstup/výstup

- `#include <stdio.h>`

  hlavičkový soubor pro standardní vstup/výstup

- `int puts(const char *s)`

  vytiskne řetězec, `char *s`, vrátí nezáporné číslo, pokud se povedlo, nebo EOF

- `int putchar(int c)`

  vytiskne jeden znak

  vrátí zapsaný znak nebo EOF

- `int getchar()`

  přečte jeden znak ze standardního vstupu, nebo EOF (konec souboru)

Terminálový vstup/výstup

- formátovaný výstup (např. čísla)
- `#include <stdio.h>`
- `int printf(const char *fmt, …)`

  vytiskne **`fmt`**, nahradí v něm specíální podřetězce (začínající znakem %) dalšími argumenty

  argumentů může být libovolně mnoho

  např.: `int a = 10; printf("Cislo je: %d\n", a);` vytiskne „Cislo je 10“

  vrací počet vytisknutých znaků (bez ukončovacího znaku `'\0'`)

  int: %d, string: %s, char: %c, float: %f

atoi()

- `int atoi(const char *s)`
- převede řetězec znaků na číslo (int)

```
int main(int argc, char *argv[])
{
   int i;
   if (argc <= 1) {
      printf("usage: program cislo\n");
      return 1;
   }
   i = atoi(argv[1]);
   printf("dvakrat tolik je: %d\n", i*2);
   return 0;
}
```

Příklad

```
/* program, který převede všechna písmena na velká  */
#include <stdio.h>

int main(int argc, char *argv[])
{
   int c;
   while ((c = getchar()) != EOF) {
      if (c >= 'a' && c <= 'z')
         c = c - ('a' - 'A');
      putchar(c);
   }
   return 0;
}
```

Domácí úkol

- sehnat a nainstalovat si kompilátor
- naučit se v něm pracovat (vytvořit projekt, kompilovat, spustit program)
- zapnout všechny warningy
- zkompilovat hello world!