Programovací jazyk C(++)

3. Soubory, pole, ukazatele

Miroslav Spousta, 2004

```
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;
        }
```

C++

Soubory

- V C: souborové operace **`fopen(), fread(), fclose, ...`**
- používá se struktura **`FILE`**

  definovaná v souboru **`stdio.h`**

  struktura: obdoba record v Pascalu

  obsah struktury je implementačně závislý, pracuje se s ní výhradně pomocí funkcí
- soubor se musí před používáním otevřít, po použití uzavřít

  kromě standardního vstupu, výstupu a chybového výstupu, jsou otevřené po spuštění

  ```
  FILE *stdin, *stdout, *stderr;
  ```
- v C od roku 1995 a C++ je možné použít wide-chars (Unicode)
- V C++ se používají proudy (streams)

Otevření souboru

- `FILE *fopen(const char *filename, const char *mode)`
- vrací ukazatel na otevřený soubor nebo 0 (NULL)

`"r"`	otevře soubor pouze pro čtení
`"w"`	pouze pro zápis, zkrátí se na nulu, pokud už existuje
`"a"`	připisování do souboru, vytvoří, pokud neexistuje
`"r+"`	otevře soubor pro vstup i výstup
`"w+"`	otevře soubor pro vstup i výstup, vytvoří nebo zkrátí
`"a+"`	čtení a připisování do souboru, vytvoří, pokud neexistuje
`"b"`	připojuje se za předchozí, značí binární mód

Zápis do textového souboru

- **`int fprintf(FILE *f, const char *fmt, …)`**

  jako **`printf`**, ale zapisuje se do souboru

  vrací počet zapsaných znaků nebo zápornou hodnotu

- **`int fputc(int c, FILE *f)`**

  vytiskne znak **`c`**, vrátí **`c`** nebo **`EOF`**

- **`int fputs(const char *s, FILE *f)`**

  nepřidává **`'\n'`** na konec (na rozdíl od **`puts`**)

  vrací poslední zapsaný znak nebo **`EOF`**

Čtení z textového souboru

- **`int fscanf(FILE *f, const char *fmt, …)`**

  jako **`scanf`**, ale čte ze do souboru

  formátovací řetězec odpovídá řádku, který chceme číst, proměnné se udávejí podobně jako v **`printf`** – uvozené znakem %

  vrací počet přečtených polí nebo **`EOF`**

- **`int fgetc(FILE *f)`**

  přečte znak, vrátí znak nebo **`EOF`**

- **`int fgets(char *s, int n, FILE *f)`**

  přečte maximálně n-1 znaků do pole **`s`**, vždy doplní **`'\0'`**

  v případě úspěchu vrací 0, jinak nenulové číslo

Binární soubory

- mohou být efektivnější než textové, neprovádí se žádné konverze
- **`size_t fread(void *data, size_t vel, size_t kolik, FILE *f)`**

  přečte na adresu **`data`**, **`vel*kolik`** bajtů ze streamu **`f`**

  vlastně čte pole o **`kolik`** prvcích, každý prvek má velikost **`vel`**

  vrací počet úspěšně přečtených prvků (**ne** bajtů)
- **`size_t fwrite(void *data, size_t vel, size_t kolik, FILE *f)`**

  zapíše z adresy **`data`**, **`vel*kolik`** bajtů do streamu **`f`**

  vrací počet úspěšně zapsaných prvků (**ne** bajtů)

Pohyb v binárním souboru

- pro každou strukturu **`FILE`** si systém udržuje aktuální pozici v souboru

  počáteční nastavení se liší podle módu otevření souboru (na začátku nebo na konci souboru)

- pozice udává, odkud se bude číst (zapisovat) při dalším čtení (zápisu)
- po úspěšně provedené operaci se ukazatel aktualizuje
- může se nastavovat i „ručně“: funkce **`ftell, fseek`**
- **`long ftell(FILE *f)`**

  pro binární soubory vrátí počet bajtů od začátku souboru

  pro textové vrátí hodnotu závislou na implementaci, která se dá použít pro **`fseek`**

Pohyb v binárním souboru

- **`int fseek(FILE *f, long okolik, int jak)`**

  nastaví aktuální pozici pro **`f`** o **`okolik`** směrem **`jak`**

  `jak`: **`SEEK_SET`** (od počátku souboru), **`SEEK_CUR`** (od aktuální pozice), **`SEEK_END`** (od konce souboru)

  vrací 0 pokud se povede, jinak nenulovou hodnotu

- mezi voláním **`fread()`** a **`fwrite()`** na stejném souboru je nutné zavolat aspoň jednou **`fseek()`**

- **`void rewind(FILE *f)`**

  nastavení pozice na začátek souboru

- **`fgetpos(), fsetpos()`**

Příklad: cat

```
#include <stdio.h>
int main(int argc, char *argv[])
{
        FILE *f; char s[100]; int n;
        if (argc <= 1) return 1;
        f = fopen(argv[1], "r");
        if (!f) return 2;
        while ((n = fread(s, 1, 100, f)) > 0) {
                fwrite(s, 1, n, stdout);
        }
        fclose(f);
        return 0;
}
```

Opakování: ukazatele

- každá proměnná má adresu (to, kde leží v paměti) a hodnotu (obsah)

  adresu proměnné získáme operátorem **`&`**

  např. **`int i; f(&i);`**

  zavolání funkce s adresou proměnné typu **`int`**

  hodnotu proměnné získáme uvedením jména proměnné

  např. **`int i, j; i = 3; j = i;`**

- ukazatel je proměnná, která má jako hodnotu adresu jiné proměnné

  je daný typ proměnné, např. ukazatel na **`int`**: **`int *p;`**

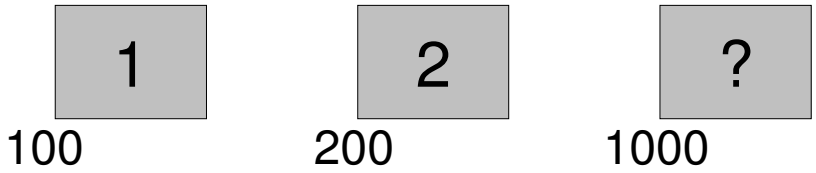
  hodnotu ukazatele získáme uvedením jména (hodnota je adresa proměnné)

  obsah odkazované proměnné získáme operátorem **`*`**

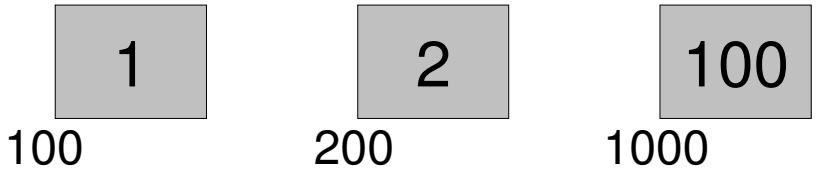
  např. **`int *p; p = &i; i = 3; (*p)++;`**

Ukazatele

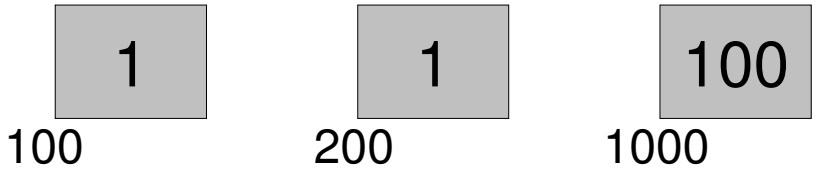
```
int x = 1, y = 2;
int *p;
```



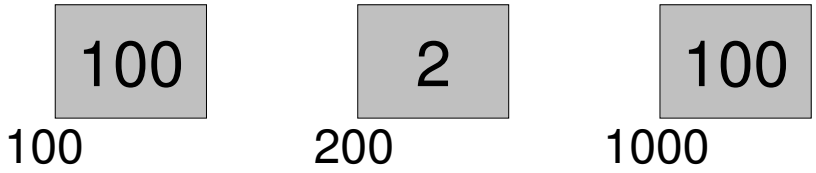

```
p = &x;
```



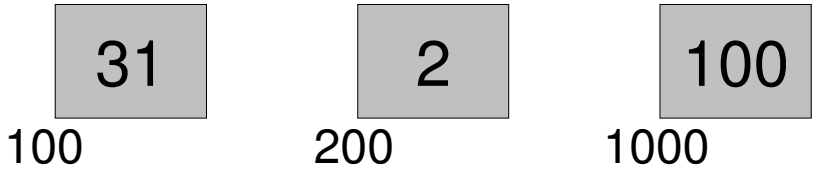

```
y = *p;
```




```
x = ip;
```




```
*p = 31;
```



Opakování: pole

- pole: několik proměnných stejného typu za sebou v paměti

  např. `int i[10]; i[0] = 3;`

- pole je v podstatě ukazatel na svůj první prvek

  ne při použití operátoru **`sizeof`** a operátoru **`&`** na celé pole

  vrátí velikost pole, resp. vrátí ukazatel na celé pole

  jinak se chová jako ukazatel na první prvek **`&x[0]`**

  pole se překládá do pointerové aritmetiky:

  `x[i]` se přeloží jako **`*(p + i)`**

  ```
  int a[10]; int *p = a; a[0] = 1; *(p+1) = *p;
  ```

  BTW: operace + je komutativní: **`x[1]`** je stejné jako **`1[x]`**

Vícerozměrná pole

- pole n-té úrovně: prvky pole jsou zase pole

  např. `int a[3][4]; i[0][2] = 3;`

- máme-li dvourozměrné `double b[2][3]`, pak `b[i][j]` se vyhodnocuje takto:

  operátor `[]` je asociativní zleva doprava: `((b[i])[j])`

  `b[i]` je totéž jako `*(b + i)` a `b[i][j]` je stejné jako `*(*(b + i) + j)`

  `b` je ukazatel na první prvek, `b + i` na i-tý prvek, typu „pole double ze tří prvků“

  `*(b + i)` je pole, převede se na ukazatel na první prvek, což je adresa prvního čísla i-té řádky

  k ní se přičte `j` (pointerová aritmetika) a dereferencí dostanu hodnotu `b[i][j]`