

Výtah z Učebnice jazyka C

978-80-7232-383-8

bodik@{civ.zcu.cz,4safety.cz}



DAY 1&2 – Základy

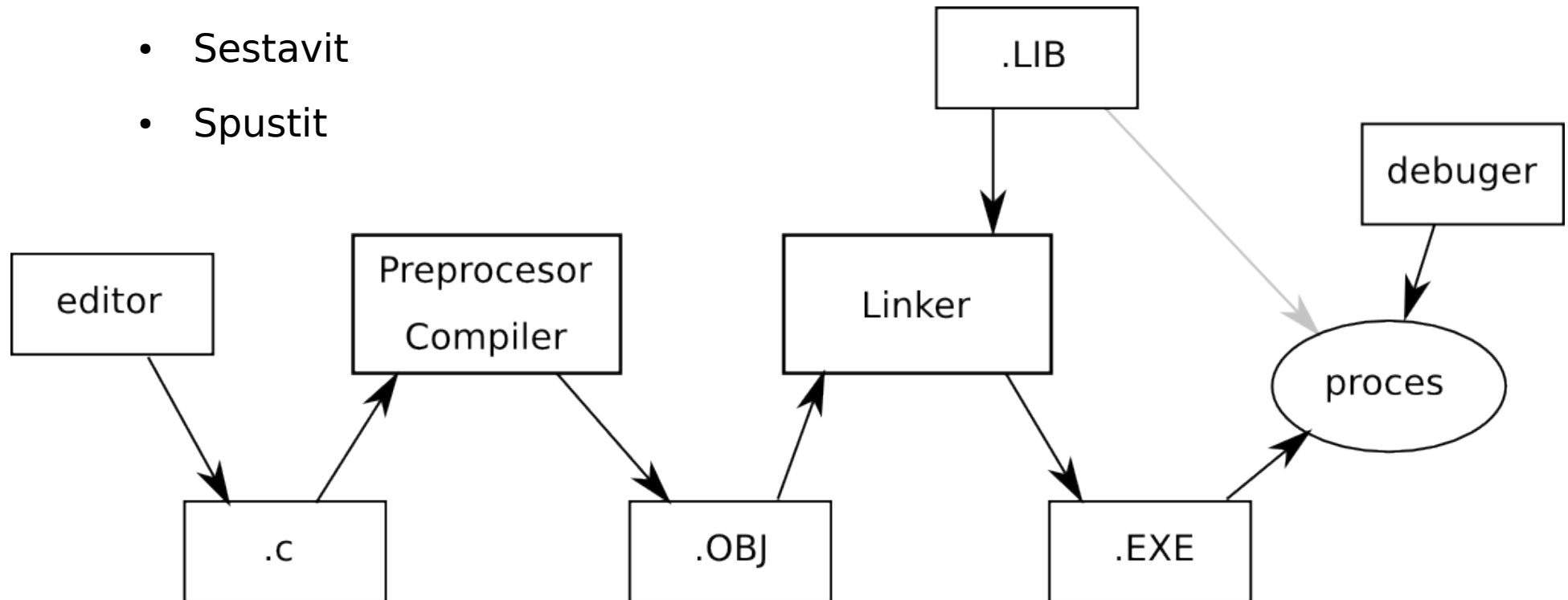
- P. Herout: Učebnice jazyka C
- Příklady z učebnice
- Různé ostatní dovednosti

1. Úvod

- Programovací jazyk (1978)
 - Brian W. Keringham a Denis M. Ritchie
 - Lowlevel > univerzální
 - Přímá práce pouze se základními datovými typy
 - To umožňuje oddělení strojově závislých implementací od jazyka do knihoven (nebo libc)
 - lit/big endian, mm, procesy, io, ...
 - Pro každý počítač je pak potřeba vytvořit příslušný překladač
- ANSI C (88)
- ISO/IEC 9899:1999
 - kosmetické úpravy, malá podpora
- *Pavel Herout*: Učebnice Jazyka C
 - ISBN 978-80-7232-383-8

2. Zpracování programu v C

- Vytvořit
- Přeložit
- Sestavit
- Spustit



Štábní kultura

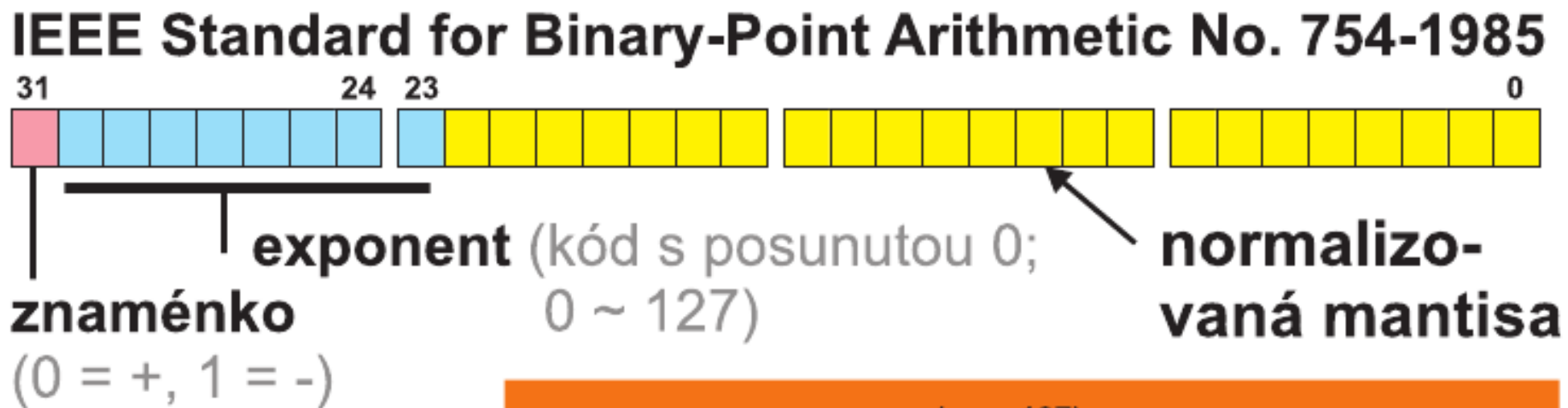
- Soubory - zdrojové (c,cpp), hlavičkové (h,hpp)
- Identifikátory
 - Casesensitive
 - Podtržítko
 - Lze použít k dělení „názvů“
 - _pom – NE (systémové věci)
 - pom_x,pBuf,p_buf,p_Buf
 - Komentáře
 - /* toto je komentář */
 - // toto není komentář ansi (c99)

3. První začátky

- Jednoduché datové typy
 - int, short int, long int <1b,8b>
 - Znaménkové $< -2^{(n-1)} ; 2^{(n-1)} - 1 >$
 - Neznaménkové $< 0 ; 2^{(n)} - 1 >$
 - char (snad skutečně 1byte)
 - wchar
 - C nemá boolean (nahradit int)
 - operátor *sizeof* – vrací velikost typu

Float a jeho zobrazení

- Přesnost na 64ti bitech není nekonečný
 - Celé číslo $\text{sum}(2^{x2}, 2^{x1}, 2^{x0})$
 - Desetinná část $\text{sum}(2^{-1}, 2^{-2}, 2^{-3})$
- Cvičení 4.7



$$h_{\text{float}} = (-1)^s \times 2^{(\text{exp} - 127)} \times (1 + \textit{mantisa})$$

Velikosti základních datových typů

Linux zzz 2.6.30serv #1 SMP x86_64 GNU/Linux

char	1
short	2
int	4
long	8
float	4
double	8
long double	16

Windows 7 Enterprise 32b

char	1
short	2
int	4
long	4
float	4
double	8
long double	8

SunOS xxx 5.9 Generic_118558-08 sun4u sparc SUNW

char	1
short	2
int	4
long	4
float	4
double	8
long double	16

Linux yyy 2.6.26-2-686 #1 SMP i686 GNU/Linux

char	1
short	2
int	4
long	4
float	4
double	8
long double	12

Proměnné

- Deklarace – svázání jména a typu
 - `extern int j;`
- Definice – svázání jména a paměti
 - `int i;`
- Přiřazení `i = 5;` (porovnání `==` !!!)
- Konstatny
 - Desítkové – 6 12
 - Osmičkově – 065 015 01
 - Hexa – 0x3A 0xCD 0x01
 - Znakové -- 'a'

Aritmetika a operátory

- Unární + -
- Binární + - * / (reálné) / (celoč.) % (zbytek)
- Speciální ++ --
- Přiřazovací += -= *= ... >>= &= |=

4. Terminálový V/V

- i/o nejsou součástí jazyka, dovoluje to ponechat práci knihovně, která může být pro každou architekturu jinak implementovaná (oddělení strojově závislých částí)
 - libc, glibc, ...
 - Znakově -- putchar(int), int getchar() (EOF)
 - Formátově – printf(), scanf()
 - Formátovací řetězec, **str 42**
 - %c %d %u %02x %f
 - Řádkově – fgets(), fputs()
- Defaultně řádkově bufferován ? Čeká se na \n
 - Bílé znaky se zahazují, přebývající zůstávají v bufferu ...

Základní program – cv4_13

- scanf, printf, scitani, deleni

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

```
int main(void)
```

```
{
```

```
    double x, y, z;
```

```
    printf("Zadej 3 realna cisla: ");
```

```
    scanf("%lf %lf %lf", &x, &y, &z);
```

```
    printf("Aritmeticky prumer je: %.30f\n", (x + y + z) / 3);
```

```
    return 0;
```

```
}
```

Základní program - cv4_13

- `scanf`, `printf`, `scitani`, `deleni` (vs `/usr/bin/bc`)

[illegible]

5. Řídící struktury

x	y	&&		XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- Rozhodování > Booleovské výrazy
 - `== != && || ! < <= > >=`
 - Priorita **str 49**
 - Zkrácené vyhodnocování podmínek
 - `if (y!=0 && z = x/y) ...`
 - `&&` vs `&` `||` vs `|`
 - `1 && 4 > 1` `1 & 4 > 0`
- Podmíněný výraz
 - `podmínka ? vyrazT : vyrazF;`

Větvení a iterace

- if, if-else
 - Přiřazení je výraz !
 - `if(a = 5)` vs `if(a == 5)`
- while
- do - while
- for
- switch
- goto (hell)
- return

Generování čísel

```
#include <stdio.h>

int main(void)
{
    double f, g;
    int i,
        d,
        h;

    printf("Zadej dve realna kladna cisla: ");
    scanf("%lf %lf", &f, &g);

    /* nalezeni horni a dolni meze */
    d = (f < g) ? f : g;
    h = (f >= g) ? f : g;

    /* zaokrouhleni na nejblizsi vyssi cele sude cislo */
    d += (d % 2) ? 1 : 0;

    for (i = d; i <= h; i += 2)
        printf("%4d ", i);

    printf("\n");
    return 0;
}
```

Vypíše sudá čísla ze zadaného intervalu

```
#include <stdio.h>

int main(void)
{
    int c,
        mala = 0,
        velka = 0;

    printf("Zadavej znaky a zadavani ukonci <Enter>\n");

    while ((c = getchar()) != '\n') {
        if (c >= 'A' && c <= 'Z')
            velka++;
        if (c >= 'a' && c <= 'z')
            mala++;
    }

    printf("\nBylo precteno:\n %d malych pismen\n %d velkych pismen\n",
        mala, velka);
    return 0;
}
```

Spočítá počty velkých a malých písmen

6. V/V souborů

- Bufferované I/O (`setvbuf()`)
- `FILE *fr`, `fopen`, `fclose`, `fgetc`, `fputc`, `fscanf`, `fprintf`
- `fgets()`
 - Interpretuje konce řádků `'\n'` (CR LF, CRLF)
 - EOF vs `feof()`;
- Defaultně otevřené proudy
 - `stdin`, `stdout`, `stderr`
- `ungetc`

V/V příklady

```
int main(void)
{
    FILE *fr1, *fr2;
    long rozdily = 0;          /* pocet rozdilnych znaku v souborech */

    if ((fr1 = fopen("PISMENA1.TXT", "r")) == NULL) {
        printf("Soubor PISMENA1.TXT se nepodarilo otevrit\n");
        return 1;
    }
    if ((fr2 = fopen("PISMENA2.TXT", "r")) == NULL) {
        printf("Soubor PISMENA2.TXT se nepodarilo otevrit\n");
        return 1;
    }

    /* test konce obou souboru pomoci feofO je zde vyhodne */
    while (feof(fr1) == 0 || feof(fr2) == 0) {
        if (getc(fr1) != getc(fr2)) {
            rozdily++;
        }
    }

    if (rozdily == 0)
        printf("Soubory jsou shodne\n");
    else
        printf("Soubory se lisi v %ld znacich\n", rozdily);

    if (fclose(fr1) == EOF)
        printf("Soubor PISMENA1.TXT se nepodarilo uzavrit\n");
    if (fclose(fr2) == EOF)
        printf("Soubor PISMENA2.TXT se nepodarilo uzavrit\n");

    return 0;
}
```

```
#include <stdio.h>

int main(void)
{
    FILE *fr;
    int i;

    if ((fr = fopen("CISLA.TXT", "w")) == NULL) {
        printf("Soubor CISLA.TXT se nepodarilo otevrit\n");
        return 1;
    }

    for (i = 1; i <= 20; i++)
        fprintf(fr, "  $%.2f\n", i * 3.14);

    if (fclose(fr) == EOF)
        printf("Soubor CISLA.TXT se nepodarilo uzavrit\n");

    return 0;
}
```

Vygeneruje 20 násobků PI na 2 desetinná místa

Spočítá počet rozdílných bytů

7. Typová konverze

- Implicitní, explicitní
- Je nutné mít na paměti že konverze probíhají a je třeba s nimi počítat
- **str 87**

```
char c; int i; double g;
```

```
i = 'A'; // konverze na int (výsledek je ord)
```

```
i = 3.8; // i -> 3
```

```
i = g * c;
```

- `c > int > double`
- `double * double > double`
- `prirazeni > int`

8. Preprocesor jazyka C

- *bin/cpp*
- Zpracovává makra
 - `#define MAXBUF 1`
 - `#include <stdio.h>`
 - `#include <moje.h>`
- Vyhazuje komentáře
- Připravuje podmíněný překlad
 - `#ifdef`
 - `#if defined`

Ukázky cpp

```
#include <stdio.h>

#define HORNI_MEZ 127

#define lze_tisknout(c) (((c) >= ' ' && (c) < HORNI_MEZ) ? 1 : 0)

int main(void)
{
    int i;

#ifdef VERBOSE
    printf("ASCII tabulka :\n");
    printf("=====\n\n");
#endif

    for (i = 0; i < HORNI_MEZ; i++) {
        if (lze_tisknout(i))
            printf("%3d %c  ", i, i);
        else
            printf("%3d -?- ", i);
    }
    putchar('\n');
    return 0;
}
```

9. Funkce a práce s pamětí

- Alokace paměti

- Vždy inicializovat ručně

- Statická

- V datové oblasti pokud je známa velikost dat při překladu
 - globální proměnné, konstanty, ...

- Dynamická

- Na haldě (heap)
 - Zásobníku (stack)
 - Lokální proměnné funkcí

- Typové modifikátory

- Const (!= Static)
 - Volatile – prom. je modifikována zvenčí

- Paměťové třídy

- Auto – lokální prom.
 - Extern – globalní prom.
 - Static
 - Glob – s omezenou viditelností
 - Local – zachovávají hodnotu
 - Register – někdy na CPU

Funkce a prototypy

- Deklarace vs definice
 - modifikátor `static` – viditelnost na lokální objekt
- Parametry volání jsou vždy předávány jako hodnota
- V C se vždy vrací hodnota (žádné procedury)
 - `void swap(int a, int b);` – návratová hodnota nás nezajímá

```
void swap(int a, int b) {  
    int p;  
    p=a; b=a; a=p;  
  
    /* vysledek vsak neni zadny, dej se odehraje  
    pouze na lokalnim stacku funkce */  
}
```

Oddělený překlad a moduly

- Paměťové třídy určují viditelnost proměnných a dat
 - auto, static, extern, ...
- Je vhodné program členit do logických celků
 - Ne všechny moduly potřebují vědět vše o zbytku programu, využívat všechny globální proměnné a nepoužívají všechny datové typy
- Program je typicky rozdělen na části/moduly, které exportují (zpřístupňují) do zbytku programu pouze části svých dat
 - Hlavičkové soubory s deklaracemi

Oddělený překlad 1

- Pokud ovládáme rozložení programu do modulů a umíme rozložit potřebné kusy kódu a paměti můžeme využívat odděleného překladu jednotlivých modulů (str. 127 – 130 – 140)

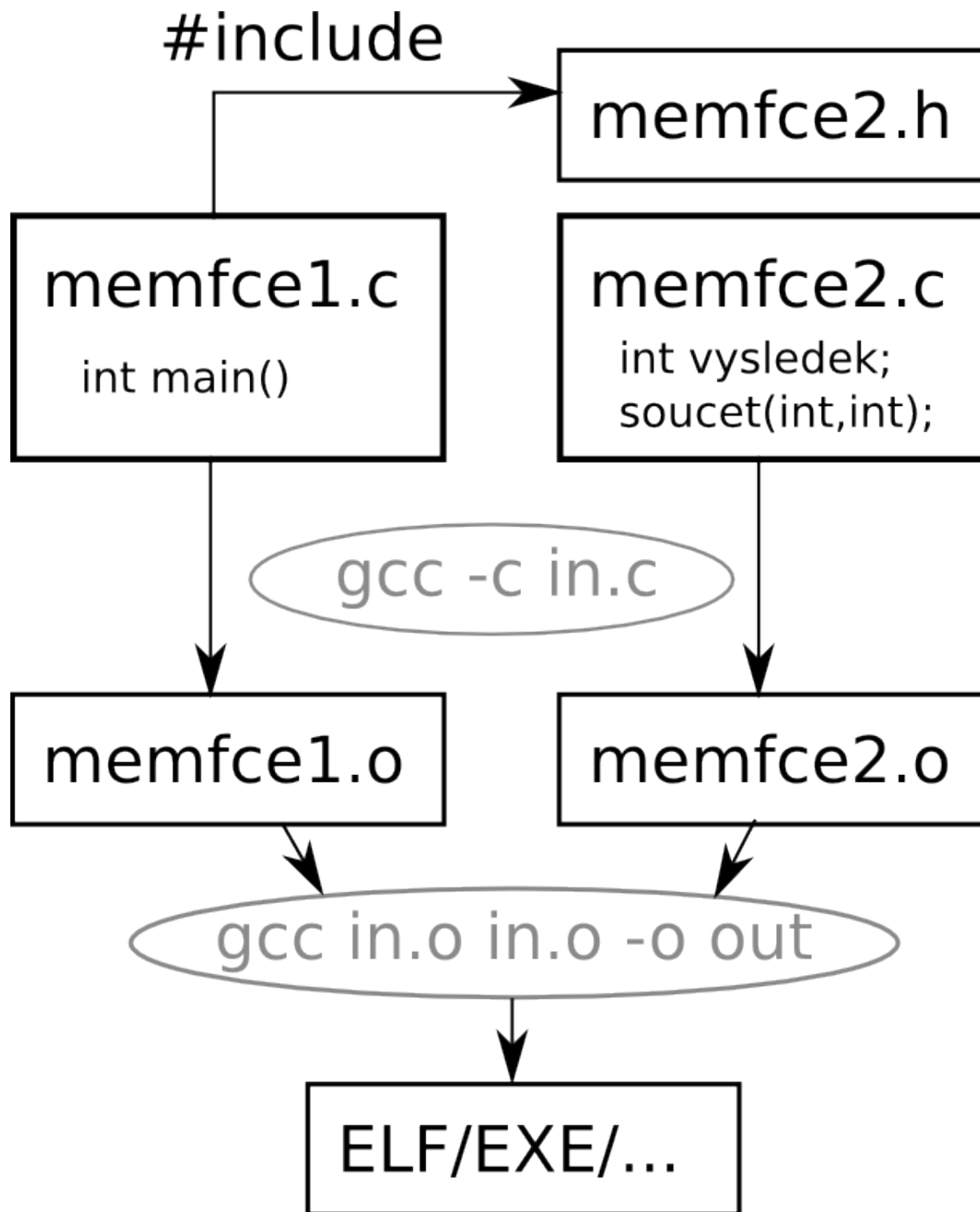
```
/* memfce2.c */  
  
#include <stdio.h>  
  
#include "memfce2.h"  
  
int vysledek;  
  
void soucet(int a, int b) {  
    static int x=0;  
  
    printf("soucet() zavolana po %d\\n",x);  
    vysledek = a+b;  
    x++;  
    return;  
}
```

```
/* memfce2.h */  
  
extern int vysledek;  
  
void soucet(int a, int b);
```

Oddělený překlad 2

```
/*  
 * memfce1.c  
 * gcc -c memfce2.c  
 * gcc -c memfce1.c  
 * gcc memfce2.o memfce1.o -o memfce  
 *  
 */  
  
#include <stdio.h>  
  
#include "memfce2.h"  
  
static int b = 1;  
  
int main(int argc, char** argv) {  
    int a = 1;  
  
    soucet(a,b);  
    soucet(a,b);  
    soucet(a,b);  
    printf("Vysledek souctu %lf\n", vysledek); //conv!  
  
    return (0);  
}
```

Oddělený překlad 3



Rekapitulace

- Jazyk C
- Štábní kultura
- Zobrazení dat (int vs float)
- Proměnné
- Řídící struktury
- Terminálový V/V
- Preprocesor
- Funkce
- Rozdělení programu do modulů a oddělený překlad

Cvičení Kap4 – term io

- Cv2, 3, 4, 8 (odečítání + konverze)
- Cv6 – hmm ... raději závorkujte a testujte
- Cv5,7 – typová konverze a zaokrouhlování
 - 3.4 vs 3.5
- Cv10,11 – magie kodování
 - *viz kit://ostatni/zobrazeni_cisel...*
- Cv12

Cvičení Kap5 – if for while

- Cv2, Cv3
- Cv4 – převod hodnot
- Cv7, Cv11, Cv14, Cv15

Cvičení Kap6 – file io

- Cv1, 2, 3, 4, 6, 7
- Cv9a vs 9b – reakce na chyby
- Cv11

- Zkuste /bin/wc
 - EOF vs feof() ?
 - Bílé znaky() ?
- Zkuste rot(x)
 - – Caesarovskou šifru, co takhle 13 ? ;)

Cvičení Kap8 – cpp

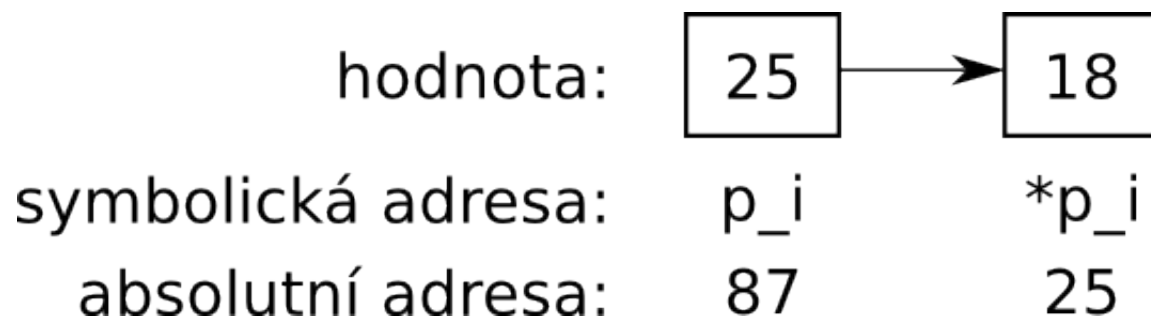
- Cvičení Cv2, 4, 6, 10

Cvičení Kap09 – fce a paměť

- Cv1, 2, 4, 5, 6, 7, 14
- Cv15 a 16 – vyzkoušení lokality
- Cv18

10. Pointery

- Ukazatel, směřník
 - Proměnná obsahující adresu skutečného obsahu
 - * .. dereferenční operátor (získává obsah)
 - & .. referenční operátor (získává adresu)
 - %p .. formátovací znak (tiskne adresu)



```
#include <stdio.h>

int main(void)
{
    int i, j, *p_i;

    scanf("%d %d", &i, &j);
    p_i = (i > j) ? &i : &j;
    printf("Vetsi je %d\n", *p_i);
    return 0;
}
```

Pointer jako parametr fce

- `void vymen(int x, int y) {`
 - Funkce nemůže změnit skutečné parametry které leží *vně fce* ...
x a y jsou lokální proměnné fce `vymen()` .. volání hodnotou
- `void vymen(int *x, int *y) {`
 - `int pom = *x;`
 - `*x = *y;`
 - `*y=pom;`
 - `}`
 - Funkce může změnit obsah na který ukazují pointery x a y
 - `scanf...`
 - Tedy „volání odkazem“ (str 152)
 - `sizeof(double *) != sizeof(double)`

Pointerová aritmetika 1

- Operátor `sizeof`

```
double d, *p_d;  
d = sizeof(d);           //velikost pro ulozeni dbl (8)  
d = sizeof(p_d);         //velikost pro ulozeni pointeru na dbl (4)  
d = sizeof(*p_d);        //velikost pro ulozeni dbl (8)
```

- součet $p+n$, rozdíl $p-n$
 - Odkaz na N tý prvek daného typu
 $p+n == p + (\text{sizeof}(*p) * n)$
- Porovnání
 - $< <= > \dots !=$
 - Má smysl pouze pokud oba pointery ukazují do stejných dat

Pointerová aritmetika 2

- Rozdíl pointerů p2-p1
 - Počet položek
$$n = p2 - p1 \rightarrow ((\text{char} *) p2 - (\text{char} *) p1) / \text{sizeof}(p2);$$
- Sčítání
 - Nelze, vyšel by nesmysl

Dynamická paměť

- Obecně tedy pointer ukazuje na paměť. Znalost pointerové aritmetiky s ním skákat, ale >> dynamické pole (ale to trochu předbíháme)
- `malloc(unsigned int)`
 - Alokuje počet bytů (někdy více podle zarovnání)
 - `calloc(unsigned int)`
 - `realloc(...)`
- `free(void *)`
 - Všechnu alokovanou paměť je potřeba uvolnit a žádnou neztrácet nevhodnou manipulací s pointery
 - Po uvolnění je vhodné nulovat ukazatel
 - `free(p); p=NULL;`

Blokové zpracování dat

Přečte 10 čísel do dynamicky alokované paměti a vypočítá součin

```
#define SIZE 3

double *init(void)
{
    return ( (double *) malloc(SIZE * sizeof(double)) );
}

void cteni(double *p_f)
{
    int i;

    for (i = 0; i < SIZE; i++) {
        printf("Zadejte %d. cislo: ", i + 1);
        scanf("%lf", p_f + i);      /* zde není & */
    }
}

void nasob(double *p_f, int size, double *p_soucin)
{ /* pozpatku, samozřejmě dopředu je to o něco snazší ;) */
    for (size--; *p_soucin = *(p_f + size); --size >= 0; )
        *p_soucin *= *(p_f + size);
}

int main(void)
{
    double *p_db1, souc;

#ifdef PLAIN
    if( ( p_db1 = (double *) malloc(SIZE * sizeof(double)) ) == NULL )
        return 1;
#else
    if ((p_db1 = init()) == NULL)
        /* nedostatek paměti - konec */
        return 1;
#endif

    cteni(p_db1);
    nasob(p_db1, SIZE, &souc);
    printf("Součin čísel je: %.3f\n", souc);
    return 0;
}
```

```

#include <stdio.h>
#include <stdlib.h>

#define SIZE 10

void init(double **p_f)
{
    *p_f = ( (double *) malloc(SIZE * sizeof(double)) );
}

void cteni(double *p_f)
{
    int i;

    for (i = 0; i < SIZE; i++) {
        printf("Zadejte %d. cislo: ", i + 1);
        scanf("%lf", p_f + i);    /* zde není & */
    }
}

void nasob(double *p_f, int size, double *p_soucin)
{
    for (size--; *p_soucin = *(p_f + size); --size >= 0; )
        *p_soucin *= *(p_f + size);
}

int main(void)
{
    double *p_dbl, souc;

    init(&p_dbl);
    if (p_dbl == NULL)
        /* nedostatek pameti - konec */
        return 1;

    cteni(p_dbl);
    nasob(p_dbl, SIZE, &souc);
    printf("Soucin cisel je: %.3f\n", souc);
    return 0;
}

```

Dynamické pole a pointer jako skutečný parametr

*Přečte 10 čísel do dynamicky alokované
paměti a vypočítá součin*

Analogicky, pointer na int, pointer
na char, tak proč ne pointer na pointer.
= je stále jenom přiřazení a po dereferenci
(aby fce mohla ovlivnit venek) musí sedět
typy operandů

Obr str 171 a příklad na str. 155

Pointer na funkce

- Pointer nemusí ukazovat jenom na data
 - Pri prekladu (c->o) predstavuje identifikator „printf“ take *pouze adresu* kde zacina funkce v uplne jinem .o (knihovne)

- `double (*p_fd)();`

- *Pointer na funkci vracející double*

```
double soucet(double, double);  
p_fd = soucet;  
w = (*p_fd)(a,b);
```

- `int (* (*f)()) [];`

- **Str 158 (188) – Jak číst komplikované definice**

- *f je pointer na funkci vracející pointer na pole prvků typu int*

- Funkce pak může být parametrem funkce (callback)

Cvičení Kap10 pointery

- Cv1,3
- Zkuste šifrování a dešifrování XORem
 - Použijte fread a fwrite (**str. 292**) kvůli problémům s \0, EOF, \r, \n

11. Jednorozměrná pole

- Pole je datová struktura složená z prvků stejného typu
 - `int pole[pocet]; // pole[0] .. pole[pocet-1]`
 - Vždy začínají od 0 {off-by-one}
 - `int pole[3]; pole[2] = 1;`
 - `pole[3] = 11; //dle zarovnání (ne)způsobí chybu`
 - C nekontroluje meze polí > ručně

Pole a pointer

- Statické pole

- `int x[10], *p_x;`

- Dynamické pole

- `p_x = (int *) malloc(10 * sizeof(int));`

- Pole a pointerová aritmetika

- `x[i] == *(p_x + i)`

- Legenda

- `x` – adresa
 - `x[1]` – hodnota
 - `&x[1]` – adresa
 - `(p_x + 1)` – adresa
 - `*(p_x + 1)` – hodnota
 - `p_x` – adresa

Práce s polem

- Nelze pracovat s celým polem najednou
 - Je třeba použít cyklus po prvcích
- Pole zvětšující svojí velikost musíme řešit sami alokací nového pole a kopií obsahu
 - malloc + kopie
 - realloc (automatická kopie)
- Pole jako parametr
 - Pole nenese informace o své délce. Pokud má figurovat jako parametr do fce, musíme navíc předávat i délku

Ukázka práce s polem

```
#include <stdio.h>

#define POCET 10

void maxim(double pole[], int pocet, double *p_max)
{
    double *p_pom;

    *p_max = pole[0];
    for (p_pom = pole + 1; p_pom < pole + pocet; p_pom++) {
        if (*p_pom > *p_max)
            *p_max = *p_pom;      /* zmena hodnoty na adrese p_max */
    }
}

int main(void)
{
    double f[POCET], max;
    int i;

    for (i = 0; i < POCET; i++) {
        printf("Zadej %d. cislo: ", i + 1);
        scanf("%lf", &f[i]);
    }
    maxim(f, POCET, &max);
    printf("Maximum z %d cislic je %f\n", POCET, max);
    return 0;
}
```

Cvičení Kap11 - arrays

- Pole pointerů na funkce a Jak číst komplikované definice
 - Str 187
- Cv1, 2, 3, 4, 5, 6

12. Řetězce

- V čistém C je řetězec jednorozměrné pole typu char
- Ukončeno znakem '\0'
 - Zde jsou lvi, offbyone, two, buffer overflow a ostatní ...

```
char s_stat[10];           //neinicializovany
char s1[10] = "ahoj";      //inicializovaný
char s2[] = "nazdar";      //inicializovaný bez udané délky
char *s3 = "ahoj";         //pointer na statickou pamet
```

```
// dynamický retezec
char *s_dyn = (char *) malloc(10 * sizeof(char));
strcpy(s_dyn, "caune");
```


Funkce pro práci s řetězci

- `scanf, fscanf, sscanf`
 - ! `scanf` pouze po první bílý znak
- `printf, fprintf, sprintf`
- `printf(const char *fmt, ...);`
 - Kompletní popis formátovacích řetězců ([str 209 - 212](#))
- `#include <string.h>`
`strlen, strcpy, strncpy (bezpečnější), strcat,`
`strchr, strrchr (práce pozpátku), strcmp, strncmp,`
`strstr, atoi, atol, atof (převod na číslo)`

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

```
int main(void)
```

```
{
```

```
    long int suma = 0;
```

```
    FILE *fr;
```

```
    int kolik;
```

```
    char akce[2];
```

```
    if ((fr = fopen("PENIZE.TXT", "r")) == NULL) {  
        printf("Soubor PENIZE.TXT nelze otevrit\n");  
        return;  
    }
```

```
    while (fscanf(fr, "%ls", akce) != EOF) {  
        fscanf(fr, " %d", &kolik);  
        /* printf("%d\n", kolik); */  
        suma += (akce[0] == '+') ? kolik : (-1 * kolik);  
    }
```

```
    printf("Celkem: $%ld\n", suma);
```

```
    if (fclose(fr) == EOF)  
        printf("Soubor PENIZE.TXT se nepodarilo uzavrit.\n");  
    return 0;  
}
```

```
$cat PENIZE.TXT
```

```
+      $10  -$5 -      $8      +$20
```

```
+$10  -      $5-$8      +$20
```

```
$█
```

Základní práce
s řetězcem

Formátované čtení z a do řetězce

- sscanf, sprintf
 - dokáže převádět převod řetězce na číslo (a to i z různých soustav)

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

```
int main(void)
```

```
{
```

```
    int i;
```

```
    char s1[5], s2[10];
```

```
    printf("Zadej 4 hexa cislice: ");
```

```
    scanf("%s", s1);
```

```
    sscanf(s1, "%x", &i);
```

```
    sprintf(s2, "%o", i);
```

```
    printf("Vysledek v osmickove soustave: %s\n", s2);
```

```
    return 0;
```

```
}
```

Řádkově orientovaný V/V

- `fputs, fgets(char *, int max, FILE *fr)`

```
#include <stdio.h>
#include <string.h>
```

```
#define MAX 5
```

```
int main(void)
{
```

```
    FILE * fr;
    char str[MAX];
    int i;
```

```
    if ((fr = fopen("cisla.txt", "r")) == NULL) {
        printf("cisla.txt neotevren\n");
        return 1;
    }
```

```
    while (fgets(str, MAX, fr) != NULL) {
        printf("' %s' : % d\t", str, strlen(str));
        for (i = 0; i < strlen(str); i++) {
            printf("%02X ", str[i]);
        }
        putchar('\n');
    }
```

```
    if (fclose(fr) == EOF) {
        printf("cisla.txt neotevren\n");
        return 1;
    }
```

```
    return 0;
```

```
}
```

- max je počet včetně ukončovací `\0`, která je doplněna, přečteno je tedy pouze max-1 znaků (viz [str 206](#))

```
$cat cisla.txt
```

```
123456789
```

```
123456789$
```

```
$/a.out
```

```
'1234' : 4      31 32 33 34
```

```
'5678' : 4      35 36 37 38
```

```
'9
```

```
' : 3 39 0D 0A
```

```
'1234' : 4      31 32 33 34
```

```
'5678' : 4      35 36 37 38
```

```
'9' : 1      39
```

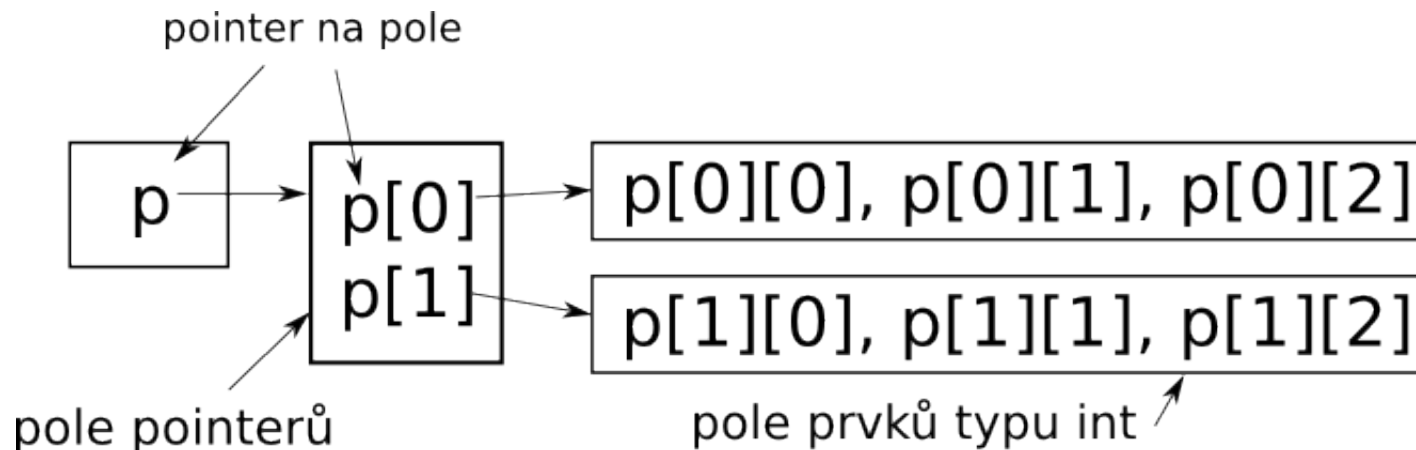
```
$
```

Cvičení Kap12 - strings

- Cv2 – kopie pomocí pointerové aritmetiky
- Cv3 – tak trochu chyták
- Cv5 – vyrobte pro převod fce(pole), které provedou příslušné konverze
- Cv7 – pointerová aritmetika
- Cv11 – velmi elegantní
- Cv13
- Cv14 – délka formátu jako argument

13. Vícerozměrné pole

- Žádná věda pro statické pole
 - `int troj[5][6][7];`
`typedef int DVA[2][3];`
`DVA x,y,z;`
 - Takto alokované pole je v paměti uloženo po řádcích `x[0][3] > x[1][0]`
- Vícerozměrné dynamické pole



Inicializace pole

```
#include <stdio.h>
#include <stdlib.h>

#define R 2
#define S 3

int a[R][S] = {1,2,3,4,5,6};

int *x[R];

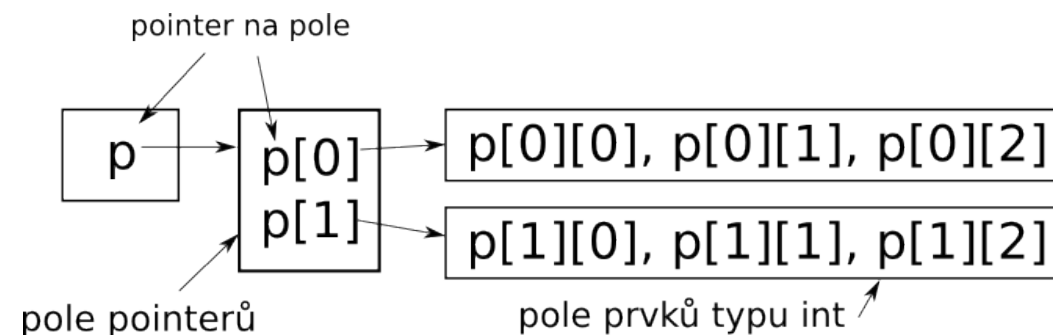
int **y, **z;

int main(int argc, char** argv) {
    x[0] = a[0];
    x[1] = a[1];

    y = (int **) malloc( R * sizeof( int * ) );
    y[0] = (int *) malloc( S * sizeof(int) );
    y[1] = (int *) malloc( S * sizeof(int) );
    y[0] = a[0];
    y[1][1] = 33;

    z = (int **) a; //nesmysl

    printf("sizeof(int)=%d\n", sizeof(int));
    printf("sizeof(int *)=%d\n", sizeof(int *));
    printf("x[1][1]:%d\n", x[1][1]);
    printf("y[0][2]:%d\n", y[0][2]);
    printf("y[1][1]:%d\n", y[1][1]);
    //printf("z[1][1]:%d\n", z[1][1]);
}
```



Nástroj pro kontrolu práce s pamětí

valgrind

- Zaměňuje volání malloc a free za obdobné funkce s počítáním

```
bodik@chronos:~/bodik-c/priklady/prezentace$ make pole
cc    pole.c    -o pole
bodik@chronos:~/bodik-c/priklady/prezentace$ valgrind ./pole
```

```
==21148== Memcheck, a memory error detector
```

```
==21148== Copyright
```

```
==21148== Using
```

```
==21148== Copyright
```

```
==21148== Using
```

```
==21148== Copyright
```

```
==21148== For more
```

```
==21148==
```

```
sizeof(int)=4
```

```
sizeof(int *)=4
```

```
x[1][1]:5
```

```
y[0][2]:3
```

```
y[1][1]:33
```

```
==21148==
```

```
==21148== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 13 from 1)
```

```
==21148== malloc/free: in use at exit: 32 bytes in 3 blocks.
```

```
==21148== malloc/free: 3 allocs, 0 frees, 32 bytes allocated.
```

```
==21148== For counts of detected errors, rerun with: -v
```

```
==21148== searching for pointers to 3 not-freed blocks.
```

```
==21148== checked 60,040 bytes.
```

```
==21148==
```

```
==21148== LEAK SUMMARY:
```

```
==21148==    definitely lost: 12 bytes in 1 blocks.
```

```
==21148==    possibly lost: 0 bytes in 0 blocks.
```

```
==21148==    still reachable: 20 bytes in 2 blocks.
```

```
==21148==    suppressed: 0 bytes in 0 blocks.
```

```
==21148== Rerun with --leak-check=full to see details of leaked memory.
```

```
bodik@chronos:~/bodik-c/priklady/prezentace$
```

definitely lost

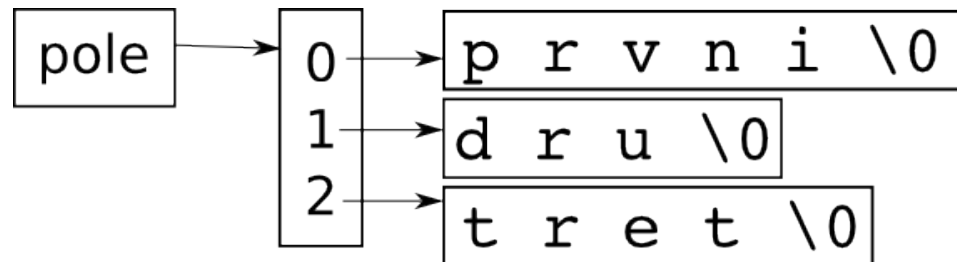
still reachable

```
y = (int **) malloc( R * sizeof( int * ) );
y[0] = (int *) malloc( S * sizeof(int) );
y[1] = (int *) malloc( S * sizeof(int) );
y[0] = a[0];
y[1][1] = 33;
```


Pole řetězců v praxi

- V praxi spíše

```
char *pole[] = {"prvni", "dru", "tret"};
```



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

```
int main(int argc, char *argv[])  
{
```

```
    int i;
```

```
    printf("Prikazova radka ma %d retezcu\n", argc);
```

```
    for (i = 0; i < argc; i++)  
        printf("%s\n", argv[i]);
```

```
    return 0;
```

```
}
```

Cvičení Kap13 - multiarrays

- Cv3,6,7,8,9,11
- Naprogramuj život
 - Data
 - Svet jako dvou rozmerne pole
 - Pozici clovicka
 - Pohyb clovicka
 - nahodny do moznych smeru
 - Zivot
 - Pohyb,Vypis sveta,Pohyb, Vypis...

14. Struktury, uniony a výčtový typ

- Struktura je heterogenní datový typ

```
typedef struct miry {           // jmeno struktury (jednotlive)
    int vyska;
    float vaha;
} MIRY, *P_MIRY;               // jmeno datovych typuu

MIRY pavel, honza;
MIRY lide[50], *p_l = lide;

pavel = honza;                 // se strukturou lze pracovat „najednou“
lide[23].vaha = 20;            // pri pristupu do struktury „.“
p_l->vyska = 3;                 // ... pres ukazatel na strukturu „->“
```

- Často se používají k tvorbě spojových seznamů
([str 237](#))

Struktury a spojový seznam

```
typedef struct peer {
    struct in_addr addr;
    int port;
    struct peer *next;
    struct peer *prev;
} PEER;

int main() {
    PEER *p_tmp, *p_all=NULL, *p_cur;
    struct in_addr start;
    struct in_addr stop;

    inet_aton("127.0.0.1", &start);
    inet_aton("127.0.0.255", &stop);

    for (; start.s_addr < stop.s_addr; start.s_addr = htonl(htonl(start.s_addr) + 1)) {

        /*pridat do seznamu*/
        p_tmp = malloc(sizeof(PEER));
        p_tmp->addr = start; /* predani cele struktury vsetne obsahu */
        p_tmp->port = SERVERPORT; p_tmp->next = NULL; p_tmp->prev = NULL;

        if (p_all == NULL) {
            p_all = p_tmp;
        } else {
            p_cur->next = p_tmp;
            p_tmp->prev = p_cur;
        }
        p_cur = p_tmp;
    }

    /* vypis vysledky */
    p_tmp = p_all;
    while (p_tmp != NULL) {
        printf("%s:%d\n", inet_ntoa(p_tmp->addr), p_tmp->port);
        p_tmp = p_tmp->next;
    }

    p_tmp = p_all;
    while (p_tmp != NULL) {
        p_all = p_tmp->next;
        if (p_all != NULL) {
            p_all->prev = NULL;
        }
        free(p_tmp);
        p_tmp = p_all;
    }
}
```

Do spojového seznamu vygeneruje IP adresy

```

typedef struct {
    char ulice[30];
    int cislo;
} ADRESA;

typedef struct {
    char jmeno[20];
    ADRESA adresa;
    float plat;
} OSOBA;

OSOBA lide[MAX]; /// oprostim

int main(void)
{
    { /* blok */
        int i, kdo = 0;
        float max = 0,
              pom;
        for (i = 0; i < MAX; i++) {
            if ((pom = lide[i].plat) > max) {
                max = pom;
                kdo = i;
            }
        }
        printf("Zamestnanec s nejvetsim platem bydlí v: %s %d\n",
              lide[kdo].adresa.ulice, lide[kdo].adresa.cislo);
    }

    { /* blok */
        float max = lide[0].plat;
        OSOBA *p_pom, *p_kdo;

        for (p_pom = p_kdo = lide; p_pom < lide + MAX; p_pom++) {
            if ((p_pom->plat) > max) {
                p_kdo = p_pom;
                max = p_pom->plat;
            }
        }
        printf("Zamestnanec s nejvetsim platem bydlí v: %s %d\n",
              p_kdo->adresa.ulice, p_kdo->adresa.cislo);
    }
    return 0;
}

```

V poli struktur najde člověka s nejvyšším platem

Vnořené struktury

Struktury, paměť a přístup

- Samozřejmě můžeme struktury, pointery a jejich dynamiku/statiku komplikovat nadevšechny meze, jen je potřeba nezapomínat na to co je přístup *přímo (.)* a *přes pointer (->)*...
 - Viz str 243 (ted')
 - používání p_ nebo jiných konvencí podporuje čitelnost
- Struktury jsou v paměti uloženy v pořadí shora dolů, ale kvůli zarovnání je potřeba vždy používat `sizeof` a vyvarovat se přístupu typu `s+offset`

Struktura a funkce 1

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

```
typedef struct {  
    double re, im;  
} KOMP;
```

```
KOMP secti(KOMP a, KOMP b)  
{  
    KOMP c;  
  
    c.re = a.re + b.re;  
    c.im = a.im + b.im;  
    return c;  
}
```

```
int main(void)  
{  
    KOMP x, y, z;  
  
    x.re = 1.1;  x.im = 3.14;  
    y = x;  
    z = secti(x, y);  
    printf("Re = %f, Im = %f\n", z.re, z.im);  
    return 0;  
}
```

- Se strukturu se pracuje jako se základními datovými typy, lze je předávat hodnotou a lze je i vracet
- nevhodné pro ty velké ...

Struktura a funkce 2

- Vhodnější je předávání pomocí pointerů

```
STUDENT *vytvor1(void) {
    STUDENT *p_pom;

    p_pom = (STUDENT *) malloc(sizeof(STUDENT));
    return p_pom;
}

void vytvor2(STUDENT **p_s) {
    *p_s = (STUDENT *) malloc(sizeof(STUDENT));
    if (*p_s == NULL)
        printf("Malo pameti\n");
}

void nastav(STUDENT *p_s, char *jmn, int rok) {
    p_s->rocnik = rok;
    strcpy(p_s->jmeno, jmn);
}

int main(void) {
    STUDENT s, *p_s1, *p_s2;

    p_s1 = vytvor1();          /* pouziti vytvor1() */
    vytvor2(&p_s2);           /* pouziti vytvor2() */

    /* prace s polozkami struktur */
    s.rocnik = 3;
    p_s1->rocnik = s.rocnik + 1;
    p_s2->rocnik = 5;
    nastav(&s, "pavel", 1);
    nastav(p_s1, "karel", 2);
    nastav(p_s2, "honza", 3);

    printf("Jmeno: %s, rocnik: %d\n", s.jmeno, s.rocnik);
    printf("Jmeno: %s, rocnik: %d\n", p_s1->jmeno, p_s1->rocnik);
    printf("Jmeno: %s, rocnik: %d\n", p_s2->jmeno, p_s2->rocnik);
    return 0;
}
```

```
typedef struct {
    char jmeno[30];
    int rocnik;
} STUDENT;
```


Výčtový typ

- Datový typ který definuje symbolické konstanty které mají vzájemnou souvislost

```
typedef enum {  
    MODRA, CERVENA, ZELENA, ZLUTA  
} BARVY;
```

```
BARVY barva = MODRA;  
char *nazvy = { "Modra", "Cervena", "Zelena" };  
  
printf("Byla to barva %s\n", nazvy[barva]);
```

Uniony

- Překryvná struktura, šetří paměť, vždy nese v daném okamžiku pouze jednu položku. Velikost přidělené paměti závisí na největším prvku
- Neobsahuje informaci o obsahu, je nutné jej znát. Často tedy ve struktuře nesoucí informaci o typu ...

```
typedef enum {  
    ZNAK, CELE, REALNE  
} TYP;
```

```
typedef union {  
    char c;  
    int i;  
    float f;  
} ZN_INT_FLT;
```

```
typedef struct {  
    TYP typ;  
    ZN_INT_FLT polozka;  
} LEPSI_UNION;
```

Cvičení Kap14

- Cv1,2,3
- Cv4 – spojový seznam
- Cv5,6,7,9

15. Bitové pole a operace

- $\&$ | \wedge -- AND OR XOR
- $\ll \gg$ -- bitový posun
- \sim -- bitová negace
- Nelze manipulovat s reálnými typy (výsledek by neměl smysl ... viz zobrazení reálných čísel v paměti)
- signed int dělají ze stejného důvodu problém se znaménkem
- Hodí se zejména ke čtení stavových slov (muhehe)

x	y	&		XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

```
int isBitSet(int x, unsigned char position) {  
    x >>= position;  
    return (x & 1) != 0;  
}
```

- Cvičení

16. Tabulka preferencí

- Hezká věc, ale raději závorkujme

17. Komplexní pohled na V/V

- Binární vs textové soubory
 - non-ascii, \0, EOF, ...
 - Z hlediska člověka je to rozdíl, z hlediska ANSI C není
- Režim otevření – to je to co C (implementaci libc) zajímá
 - Binární – nikde žádná konverze
 - Textový – manipulace s \r (konec řádky)
 - Problém nastane při nevhodné kombinaci (textově otevřeme s binárním souborem)
 - EOF(0xFF) Ctrl-Z (0x1A) Ctrl-D (0x04) \r (0x0D)

Funkce pro blokové čtení a zápis

- `fread(dest*,size,n,FILE*)`
- `fopen, fwrite, fseek, ftell, fclose`
 - **str 303 - 305** (čtení *struktur* – asi není moc přenositelné)
 - **! str 305** – operace čtení a zápis po sobě ?
- `fflush, setbuf, setvbuf`
 - Všechny V/V operace jsou bufferované OS a je to tak dobře
 - Řádkové
 - Blokové
 - Žádné
- `feof, ferror`
- `freopen, rename, tmpfile, tempnam, ...`

Ukázka práce se souborem

```
#include <stdio.h>
#include <stdlib.h>

#define DELKA_BLOKU 5

int main(void)
{
    char blok[DELKA_BLOKU];
    FILE *fr, *fw;
    int precteno;

    fr = fopen("A.TXT", "rb");
    fw = fopen("B.TXT", "wb");

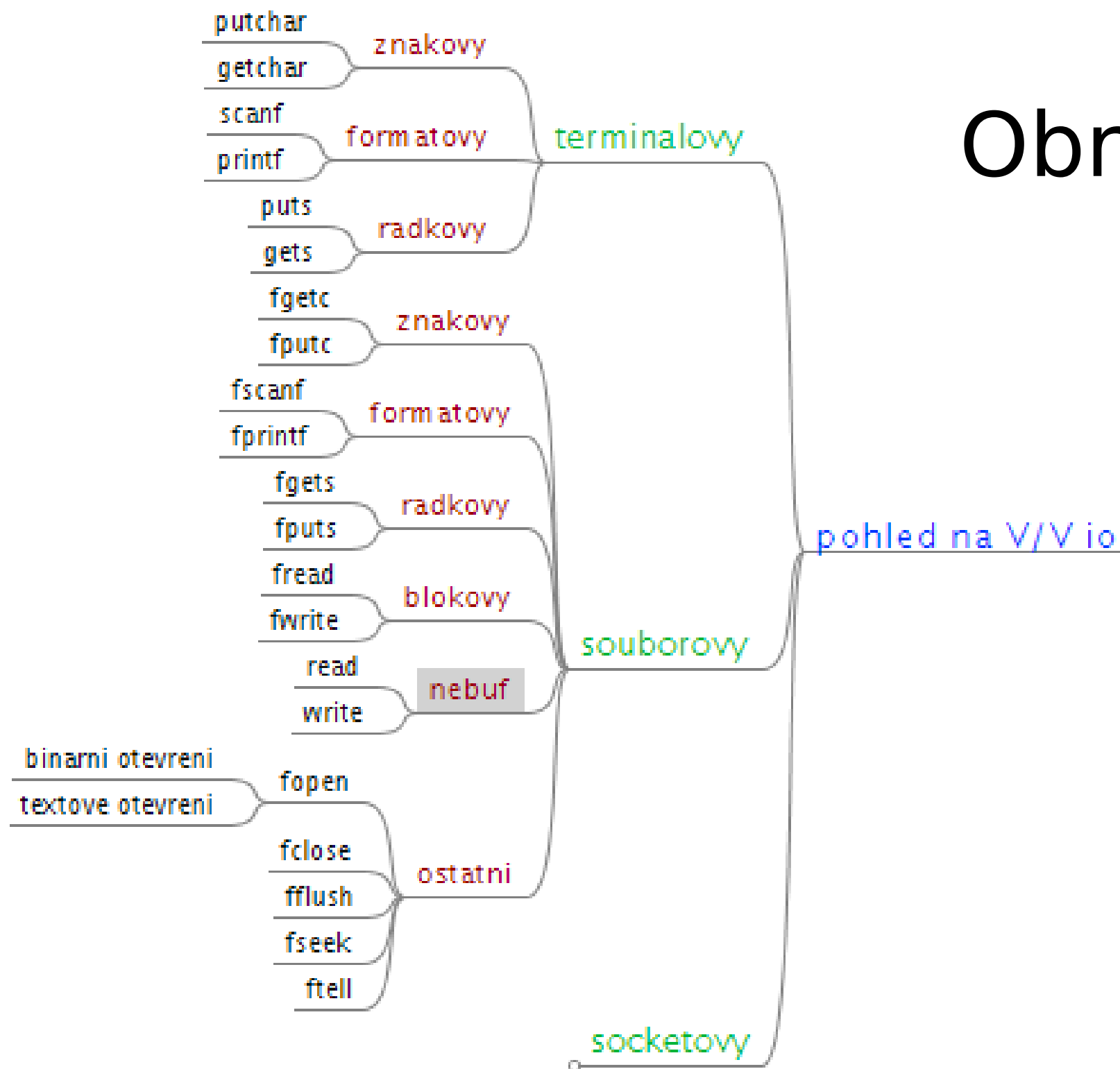
    while ((precteno = fread(blok, sizeof(char), DELKA_BLOKU, fr)) == DELKA_BLOKU) {
        if (fwrite(blok, sizeof(char), DELKA_BLOKU, fw) != DELKA_BLOKU) {
            fprintf(stderr, "Chyba pri zapisu do souboru\n");
            exit(1);
        }
    }
    /* cteni skoncil prectenim vseho nebo chybou */

    if (ferror(fr) != 0) {
        fprintf(stderr, "Chyba ve cteni souboru\n");
    } else {
        /* skoncil prectenim mene dat, nez je velikost bloku */
        /* prectena data je nutno zapsat, i kdyz jich je mene */
        if (fwrite(blok, sizeof(char), precteno, fw) != precteno)
            fprintf(stderr, "Chyba pri zapisu do souboru\n");
    }

    fclose(fr);
    fclose(fw);
    return 0;
}
```

Blokově zkopíruje soubor

Obrázek



```
#include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>
```

```
int main(int argc, char **argv) {
    DIR          *p_dir;
    struct dirent *p_dirent;
    struct stat   s_buf;
    char          plne_jmeno[1000];

    p_dir = opendir(argv[1]);
    if (p_dir == NULL) {
        printf("Adresar '%s' neexistuje\n", argv[1]);
        return 1;
    }

    printf("Podadresare:\n");
    while ((p_dirent = readdir(p_dir)) != NULL) {
        sprintf(plne_jmeno, "%s/%s", argv[1], p_dirent->d_name);
        stat(plne_jmeno, &s_buf);
        if ((s_buf.st_mode & S_IFDIR) != 0) {
            if (p_dirent->d_name[0] != '.')
                printf("%s\n", p_dirent->d_name);
        }
    }

    rewinddir(p_dir);

    printf("\nSouborny:\n");
    while ((p_dirent = readdir(p_dir)) != NULL) {
        sprintf(plne_jmeno, "%s/%s", argv[1], p_dirent->d_name);
        stat(plne_jmeno, &s_buf);
        if ((s_buf.st_mode & S_IFDIR) == 0) {
            printf("%s\n", p_dirent->d_name);
        }
    }

    return 0;
}
```

- Fce nejsou součástí ANSI C, proto je kód nepřenosný
- **str 312**

Obsah adresáře

Ostatní funkce

- Seznam základních hlavičkových souborů
 - [Str 320](#)
- `rand()` – [str 340](#)
- Vyhledávání a řazení – [str 344](#)
- `strtok()` – snad později ([str 349](#))
- `memcpy()` – bloková kopie oblasti paměti ([str 351](#))
- `time`, `localtime`, `strftime`, `mktime`
– práce s daty ([str 359](#))

19. Ladění programů

- logovat program – `printf()` / `fprintf()`
 - Zapínat nebo vypínat výpisy preprocesorem
 - Na základě argumentů/konfigurace programu
- Debuggerem – runtime vs. post mortem
 - `gdb` ... hardcore
 - `ulimit -c unlimited; gdb program core`
 - `$ back, list, break, run, edit, set print pretty on`
 - `ddd` ... krasne zobrazení
 - Netbeans .. trochu tricky, ale větší komfort
 - Nezapomenout přepínač `-g`

20. Unix a GCC

- GCC – GNU Compiler
 - 2.95 dlouholetý standard, 4.x současnost
- # prelozi a sestavi program do a.out
`gcc pokus.c`
- # dtto do pokus s kontrolou dle standardu
`gcc -ansi pokus.c -o pokus`
- # prelozi objekt s ladici symboly, navíc definuje symbolickou konstantu
`gcc -g -DDEBUG pokus.c -c`
- # sestavi program z modulu
`gcc pokus.o pokus1.o -o pokus`
- `-Wall -ansi -pedantic -g -O3`
- `-L<libdir> -l<libname> -I<includesdir>`

make

- Nástroj pro řízení a automatizaci překladu
 - Oddělený překlad modulů
 - automake, autoconf ... makrošílenství
 - Make se řídí konfiguračním skriptem a časovými razítky souborů

```
# pokusny soubor pro make
```

```
# pozadavek ladicich informaci
```

```
OPT=-g
```

```
pozdravy: hlavni.c ahoj.o nazdar.o cao.o
```

```
gcc ${OPT} ahoj.o nazdar.o cao.o hlavni.c -o $@
```

```
ahoj.o: ahoj.c ahoj.h nazdar.h cao.h max.h
```

```
gcc ${OPT} -c ahoj.c
```

```
nazdar.o: nazdar.c nazdar.h cao.h max.h
```

```
gcc ${OPT} -c nazdar.c
```

```
cao.o: cao.c cao.h max.h
```

```
gcc ${OPT} -c cao.c
```

kit://

- Potreby pro cviceni
 - install
 - Vypracovaná cvičení a ukázky z knihy
 - Další ukázky
 - ssh .. shell
 - Spojovy seznam
 - Get .. packet sniffer
 - Openficker4 .. PoC
 - Regexp in C
 - Prikklady z prezentace
 - Ruzne ostatni prikklady
 - Sockety
 - Xor file „cypher”

Co se nevešlo

- Desatero (**str 425**)
- `$ apt-get install manpages-dev gcc-doc glibc-doc`
- diff vs patch
- man, apropos
- Tvorba funkcí s proměnným počtem parametrů
 - Str 337
- lint, splint – statická kontrola zdrojových textů
- Regularni vyrazy v C (`kit://prikklady/pam...`)
- glibc
- gtk

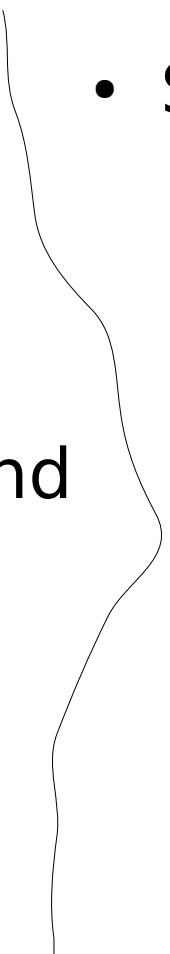
Getopt

- Zpracování argumentů programu z příkazové řádky standardizovaným způsobem
 - http://www.gnu.org/s/libc/manual/html_node/Example-of-Getopt.html

Get – packet sniffer

- `kit://prik lady/get`

Sockety

- *TODO*
 - *kit://ostatni/sockety*
 - Klient
 - struct addr, int fd
 - socket
 - connect
 - read,write || recv,send
 - close
 - Server
 - struct addr, int fs, int fc
 - socket
 - bind
 - listen
 - accept, select
 - read,write || recv,send
 - close
- 

Vláčna

- <http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html>

DAY 3

- Podle potřeby goto DAY4

DAY 4 - openficker

- viz Nebezpecnost 2009
- makefile je špatně

Výtah z Učebnice jazyka C

80-7232-115-3

bodik@{civ.zcu.cz,4safety.cz}

