KATEDRA INFORMATIKY
PŘÍRODOVĚDECKÁ FAKULTA
UNIVERZITA PALACKÉHO V OLOMOUCI

Bitové operátory, bitová pole a mnoho dalších užitečných maličkostí

10. cvičení

Jiří Zacpal

KMI/ZP2 – Základy programování 2

Bitové operátory

- & (součin), | (součet), ^ (nonekvivalence, XOR), >> (posun vpravo), << (posun vlevo), ~ (negace)
- argumenty operátorů mohou být pouze celočíselné
- vyhodnocení bitových operátorů je relativně rychlé
- příklady
  - převod na velká písmena
    ```
    c = c & 0xDF;
    ```
  - převod na malá písmena
    ```
    c |= 0x20;
    ```
  - násobení mocninou čísla 2
    ```
    x = x << 3;
    ```
  - dělení mocninou čísla 2
    ```
    x >>= 2;
    ```

Práce se skupinou bitů

- bitové operátory se často používají při práci se skupinami bitů (více informací uložených v jedné proměnné)
- příklad:

```
#define READ 0x8
#define WRITE 0x10
#define DELETE 0x20
...
unsigned int status;
status |= READ | WRITE | DELETE;
status |= READ | WRITE;
status &= ~(READ | WRITE | DELETE);
status &= ~READ;
```

Vytvoření bitového pole

- bitová pole umožňují pracovat s jednotlivými bity nebo skupinami bitů proměnných pomocí symbolických jmen
- definice bitového pole

```
typedef struct {
   unsigned den : 5;
   unsigned mesic : 4;
   unsigned rok : 7;
} DATUM;
```

- jednotlivé položky mohou být pouze celé znaménkové (signed) nebo celé neznaménkové (unsigned)
- číslo za dvojtečkou udává počet bitů pro danou položku
- velikost celého bitového pole bývá shora omezena hodnotou sizeof(int)

Práce s bitovým polem

- nelze pracovat s adresami (ani ukazateli) na položky
- s bitovým polem lze pracovat jako se strukturou
- vytvoření proměnné (s inicializací)

```
DATUM dnes = {23, 4, 2008 - 1980};
DATUM zitra = dnes;
```

- přístup k položkám

```
zitra.den++;
dnes.mesic = 6;
dnes.rok = 2009 - 1980;

if
((dnes.rok!=d.rok)||(dnes.mesic!=d.mesic)||(dn
es.den!=d.den))…

printf("%u. %u. %u\n", d.den, d.mesic,
d.rok + 1980);
```

Typový modifikátor `const`

- slouží pro označení proměnné, jejíž hodnota se po inicializaci již nebude měnit

```
const double pi = 3.14159;
const max = 100;
```

- takto vytvořené konstanty mají daný typ a rozsah platnosti v odpovídajícím bloku
- lze deklarovat i konstantní ukazatel nebo ukazatel na konstantní hodnotu

```
const static char *t1= "Pointer na const";
static char *const t2 = "Const pointer";
const static char *const t3 =
"Const pointer na const znaky.";
```

Konverze textu na číslo

- níže zmíněné funkce jsou součástí knihovny `stdlib.h`
- jednoduché převody textu na číslo

```
int atoi(const char* str);
long atol(const char* str);
double atof(const char* str);
```

- složitější převody textu na číslo

```
long strtol(const char *s, char **konec, int zak);
unsigned long strtoul(const char *s, char **konec, int
zak);
double strtod(const char *s, char **konec);
```

Funkce převádí textový řetězec s na číslo odpovídajícího typu. V parametru konec je po volání funkce vrácen ukazatel na konec zpracováného textu. Pomocí parametru zak lze zadat základ vstupní číselné soustavy (hodnoty 2 až 36).

- příklad:

```
char *k;
long int cislo, cislo2;
cislo = atol("12345");
cislo2 = strtol("AB12D", &k, 13);
```

Generování pseudonáhodných čísel

- inicializace generátoru (funkce z knihovny stdlib.h)
  ```
  void srand(unsigned int seed);
  ```
- jako seed při inicializaci často používá funkce (z time.h)
  time_t time(time_t *out);
  Tato funkce vrací počet sekund od 1. ledna 1970 do svého volání.
- generování náhodných čísel (funkce z stdlib.h)
  ```
  int rand();
  ```
  Funkce vrací číslo od 0 do `RAND_MAX`.
- generování čísel v jiném rozsahu
  ```
  cislo = rand() % (max + 1 - min) + min;
  ```
- příklad:
  ```
  int pole[20];
  int i;
  srand((unsigned int)time(NULL));
  for (i=0; i<20; i++) pole[i] = rand() % 10 + 1;
  ```

Funkce pro řízení programu

- níže zmíněné funkce jsou součástí knihovny `stdlib.h`
- funkce pro ukončení programu
  ```
  void exit(int kod);
  ```
  Funkce ukončí provádění programu, zapíše buffery a uzavře proudy, smaže dočasné soubory, volá funkce registrované pomocí funkce atexit. Parametr kod odpovídá návratové hodnotě programu (jakoby návratové hodnotě funkce main).
- registrace funkcí, které se budou spouštět při standardním ukončení programu (volání exit nebo return ve funkci main)
  ```
  int atexit(void (* funkce)());
  ```
- funkce pro nestandardní ukončení programu
  ```
  void abort();
  ```
  Funkce pro rychlé ukončení programu; nezapisuje buffery, nemaže dočasné soubory, nevolá funkce registrované pomocí atexit.
- funkce pro komunikaci s OS (příkazy OS, spouštění programů)
  ```
  int system(const char *prikaz);
  ```

Matematické funkce

- definované ve standardní knihovně `math.h`
- funkce mají vesměs intuitivní názvy, jeden vstupní parametr typu double a návratovou hodnotu typu double
  `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `sinh`, `cosh`, `tanh`, `exp`, `log`, `log10`, `sqrt`, `ceil`, `floor`, `fabs`
- některé funkce mají 2 vstupní parametry
  `pow`, `fmod`, `modf`
- příklady:

```
double x = 12.3;

double zaok_dolu = floor(x);
double zaok_nahoru = ceil(x);

double radiany = asin(sqrt(3)/2);

double zbytek = fmod(x, 2.1);
double cela_cast;
double des_cast = modf(x,&cela_cast);
```

Parametry základních datových typů

- pro lepší přenositelnost programů je vhodné mít k dispozici symbolické konstanty, které popisují rozsah a případé další paramerty jednotlivých základních typů
- tyto konstanty jsou definovány ve float.h (konstanty týkající se reálných typů) a limits.h (konstanty popisující celočíselné typy)
- jména konstant jsou opět poměrně intuitivní:
  `INT_MAX`, `INT_MIN`, `UINT_MAX`, `SHRT_MAX`, **...** ,
  `DBL_MIN`, `DBL_MAX`, `DBL_EPSILON`, `DBL_DIG`, **...**
- obsahy hlavičkových souborů si můžete v případě potřeby sami prohlédnout, většinou jsou dobře okomentované

Standardní makra pro práci se znaky

- definována ve standardní knihovně `ctype.h`
- názvy maker jsou opět velmi intuitivní
  `isalpha`, `isalnum`, `isdigit`, `isxdigit`, `islower`, `isupper`, `isprint`, `ispunct`, `isgraph`, `isspace`, `iscntrl`, `tolower`, `toupper`
- příklad:

```
int c, i;
int pismena[26];
FILE *fr = fopen("text.txt", "r");

for (i=0; i<26; i++) pismena[i] = 0;

while ((c=getc(fr))!=EOF){
    if (isalpha(c)) {
            if (islower(c)) c = toupper(c);
            pismena[c-'A']++;
    }
}
```

Práce s textovými řetězci 1/2

- většina funkcí pro práci s textem je obsažena ve `string.h`
- formátované čtení a zápis textových řetězců (`stdio.h`)

```
int sscanf(const char * zdroj, const char *format, ...);
int sprintf(char* cil, const char* format, ...);
```

- zatím nezmíněné funkce ze `string.h`

```
size_t strspn(const char *zdroj, const char *set);
```

Funkce vrací počet prvních znaků řetězce zdroj, které **jsou** všechny obsaženy ve množině znaků `set`.

```
size_t strcspn(const char *zdroj, const char *set);
```

Funkce vrací počet prvních znaků řetězce zdroj, které **nejsou** (všechny) obsaženy ve množině znaků `set`.

```
char *strpbrk(char *zdroj, const char *set);
```

Funkce vrací ukazatel na první znak zdroj, který je obsažený ve množině znaků `set`.

```
char *strtok(char *zdroj, const char *set);
```

Funkce pro rozdělnení řetězce zdroj oddělovači z množiny set. Při prvním volání se předá zdroj, při dalších se předává NULL.

Práce s textovými řetězci 2/2

- funkce pro práci s blokem bytů (znaků), který není ukončen `'\0'`

  ```
  void *memchr(void *p, int h, size_t v);
  ```
  Funkce vyhledá v paměti p o velikosti v první byte s hodnotou h.
  V případě neúspěchu vrací NULL, jinak ukazatel na nalezený byte.
  Jedná se o obdobu funkce `strchr`.

  ```
  int memcmp(const void *p1, const void *p2, size_t v);
  ```
  Funkce pro porovnání bloků paměti, obdoba `strcmp`.

  ```
  void *memcpy(void *kam, const void *co, size_t v);
  ```
  Funkce pro kopírování bloku paměti, obdoba `strcpy`.

  ```
  void *memmove(void *kam, const void *co, size_t v);
  ```
  Jako `memcpy`, pouze se může překrývat paměť `kam` a `co`.

  ```
  void *memset(void *p, int h, size_t v);
  ```
  Vyplní prvních v bytů paměti p hodnotou h zkonvertovanou
  na unsigned char a vrátí ukazatel na paměť p.

Úkol

Prostudujte si zdrojový kód v připraveném souboru a dopište funkci DATUM maximum(char *nazev). Tato funkce by měla číst datumy (využijte bitové pole DATUM) z binárního souboru nazev a vrátit největší (nejpozdější) datum jako svou návratovou hodnotu.

Typ DATUM můžete také předefinovat jako union výše zmíněného bitového pole a neznáménkového celého čísla, čímž si můžete zjednodušit porovnávání 3 hodnot (rok, měsíc a den) na porovnávání jediné hodnoty. Pozor ovšem na pořadí bitů ve struktuře bitového pole a velikosti typů na konkrétním počítači. Toto řešení je implementačně závislé!

Příklad výstupu:

```
Nejpozdejsi datum je: 23. 4. 1995
```

Úkol – řešení

```
DATUM maximum(char *nazev)
{
     FILE *fr;
     DATUM max={0,0,0},d;
     fr = fopen(nazev, "rb");
     if (fr == NULL) return max;
     while (fread(&d, sizeof(DATUM), 1, fr)== 1)
     {
          if(d.rok>max.rok)
               max=d;
          else if((d.rok==max.rok)&&(d.mesic>max.mesic))
               max=d;
          else if((d.rok==max.rok)&&(d.mesic==max.mesic)&&(d.den>max.den))
               max=d;
       }
     fclose(fr);
     return max;
}
```

Bodovaný úkol

Prostudujte si zdrojový kód v připraveném souboru a dopište funkce

```
mnozina prunik(mnozina A, mnozina B);
mnozina sjednoceni(mnozina A, mnozina B);
mnozina rozdil(mnozina A, mnozina B);
```

pro operace s danými množinami A a B. Pro reprezentaci množin použijte připravený strukturovaný typ mnozina, jehož člen pocet odpovídá počtu možných indexů (tj. počtu prvků univerza) a člen prvky obsahuje posloupnost bitů (rozčleněnou do několika položek typu int tvořících pole), která udává, který prvek patří (bit s hodnotou 1) a který nepatří (hodnota 0) do dané množiny.