# Odvozené a strukturované typy dat

Petr Šaloun

katedra informatiky FEI VŠB-TU Ostrava

14. listopadu 2011

# Odvozené a strukturované typy dat

- základní datový typ – součástí normy jazyka,
- preprocesor – použití bezparametrických maker, raději volíme konstanty
- odvozené datové typy

  **typedef** <type definition> <identifier>;

  vlastní definice datového typu
- strukturované datové typy
  **struct** a **union**
  vlastní definice struktury.

# Složitější typové deklarace

| deklarace | význam |
|---|---|
| `typ jméno;` | `typ`, |
| `typ jméno[];` | (otevřené) pole typu `typ`, |
| `typ jméno[3];` | pole (pevné velikosti) tří položek typu `typ` (`jméno[0]`, `jméno[1]`, `jméno[2]`), |
| `typ *jméno;` | ukazatel na `typ`, |
| `typ *jméno[];` | (otevřené) pole ukazatelů na `typ`, |
| `typ *(jméno[]);` | (otevřené) pole ukazatelů na `typ`, |
| `typ (*jméno)[];` | ukazatel na (otevřené) pole typu `typ`, |
| `typ jméno();` | funkce vracející hodnotu typu `typ`, |
| `typ *jméno();` | funkce vracející ukazatel na hodnotu typu `typ`, |
| `typ *(jméno());` | funkce vracející ukazatel na hodnotu typu `typ`, |
| `typ (*jméno)();` | ukazatel na funkci, vracející `typ`. |

Obecný postup – zevnitř ven.

Postup pro správnou interpretaci definice:

1. začněme u identifikátoru a hledejme vpravo kulaté nebo hranaté závorky (jsou–li nějaké);
2. interpretujme tyto závorky a hledejme vlevo hvězdičku;
3. pokud narazíme na pravou závorku (libovolného stupně vnoření), vraťme se a aplikujme pravidla jedna a dva pro vše mezi závorkami;
4. aplikujme specifikaci typu.

# Příklad složitější typové konstrukce

```
char  *(  *(  *var)  ())  [10];
  7   6   4   2 1     3     5
```

1. identifikátor `var` je deklarován jako
2. ukazatel na
3. funkci vracející
4. ukazatel na
5. pole deseti prvků, které jsou
6. ukazateli na
7. hodnoty typu `char`.

# Příklady složitější typové konstrukce – 2

```
unsigned int *(*const *name[5][10]) (void);
```

Identifikátor `name` je dvourozměrným polem o celkem padesáti prvcích. Prvky tohoto pole jsou ukazateli na ukazatele, které jsou konstantní. Tyto konstantní ukazatele ukazují na typ funkce, která nemá argumenty a vrací ukazatel na hodnotu typu `unsigned int`.

```
double (*var (double (*)[3])) [3]
```

Funkce vrací ukazatel na pole tří hodnot typu `double`. Její argument, stejně jako návratová hodnota, je ukazatel na pole tří prvků typu `double`.

Argument funkce var je konstrukce, která se nazývá *abstraktní deklarace*. Obecně se jedná o deklaraci bez identifikátoru. Pro zjednodušení a zpřehlednění abstraktních deklarací se používá `typedef` konstrukce.

# Příklady abstraktní deklarace typů

| deklarace | význam |
|---|---|
| `int *` | ukazatel na typ `int`, |
| `int *[3]` | pole tří ukazatelů na `int`, |
| `int (*)[5]` | ukazatel na pole pěti prvků typu `int`, |
| `int *()` | fce bez specifikace arg. vracející uk. na `int`, |
| `int (*) (void)` | uk. na fci nemající argumenty vracející `int`, |
| `int (*const [])` `(unsigned int, ...)` | uk. na nespecifikovaný počet konst. uk. na fce. z nichž každá má první argument `unsigned in` a nespecifikovaný počet dalších argumentů. |

# Výčtový typ

umožňuje definovat konstanty výčtového typu.

**enum** [tag] {**enum**–list} [declarator];

- `enum` je klíčové slovo, zahajující definici hodnot výčtového typu;
- `tag` je nepovinná „visačka“, využívaná zejména ve stylu K&R jazyka C, s konstrukcí `typedef` četnost jejího použití klesá;
- `enum-list` je seznam konstant výčtového typu s možnou explicitně přiřazenou hodnotou, viz příklad dále, jinak nabývá první konstanta výčtového typu hodnoty nula, druhá hodnoty jedna, . . . , každý následník má hodnotu o jedničku vyšší, než jeho předchůdce;
- `declarator` je nepovinný seznam proměnných daného typu `enum`.

# Použití konstant výčtových typů – enum_use.c

```
typedef enum {
    Back = 8, Tab = 9, Esc = 27, Enter = 13,
    Down = 0x0150, Left = 0x014b, Right = 0x014d,
    Up = 0x0148, NUL = 0x0103, Shift_Tab = 0x010f,
    Del = 0x0153, End = 0x014f, Home = 0x0147,
    Ins = 0x0152, PgDn = 0x0151, PgUp = 0x0149
} key_t;
int znak;
 ...
 else if ((znak == Left) || (znak == Back))
 ...
 else if (znak == Enter)
 ...
 else if (znak == Esc)
 ...
 else if ...
```

# Typ struktura

```
struct [<struct-type-name>] {
   [<type> <variable-name[, variable-name, ...]>] ;
   [<type> <variable-name[, variable-name, ...]>] ;
   ...
} [<structure variables>] ;
```

. „tečka“ – selektor struktury (záznamu), např. u proměnné.

-> – selektor struktury, prostřednictvím ukazatele.

# Příklad struktur – struct01 .c – I

```
typedef
 struct {float re, im;} complex;

 complex cislo,
         im_jednotka = {0, 1};
 cislo.re = 12.3456;
 cislo.im = -987.654;
 printf("re = %10.5f im = %10.5f\n",
         im_jednotka.re, im_jednotka.im);
 printf("re = %10.5f im = %10.5f\n",
         cislo.re, cislo.im);
```

# Příklad struktur – struct01 .c – II

```
typedef
   struct {int ev_cislo;
           char nazev[ZNAKU_NAZEV + 1];
           int  na_sklade;
           float cena;
          } vyrobek;
typedef vyrobek zbozi[POLOZEK_ZBOZI];

 vyrobek a = {8765, "nazev zbozi na sklade",
              100, 123.99};
 vyrobek *ppolozky;
 ...
 ppolozky->ev_cislo = 1;
```

# Příklad struktur – struct01 .c – III

```
polozky[0].ev_cislo = 0;
strcpy(polozky[0].nazev, "polozka cislo 0");
polozky[0].na_sklade = 20;
polozky[0].cena = 45.15;

ppolozky = polozky + 1;
ppolozky->ev_cislo = 1; /* (*ppolozky).ev_cislo = 1;*
strcpy(ppolozky->nazev, "polozka cislo 1");
ppolozky->na_sklade = 123;
ppolozky->cena = 9945.15;

printf(FORMAT_VYROBEK, a.ev_cislo, a.na_sklade, a.cen
printf(FORMAT_VYROBEK, polozky[0].ev_cislo, polozky[0
       polozky[0].cena, polozky[0].nazev);
printf(FORMAT_VYROBEK, ppolozky->ev_cislo, ppolozky->
       ppolozky->cena, ppolozky->nazev);
```

# Použití struktur – výstup struct01.c

```
re =     0.00000 im =     1.00000
re =    12.34560 im =  -987.65399
cislo:  8765 pocet:  100 cena:   123.99 nazev:nazev zbozi
cislo:     0 pocet:   20 cena:    45.15 nazev:polozka cis
cislo:     1 pocet:  123 cena:  9945.15 nazev:polozka cis
```

# Příklad struktury FILE * – datové proudy

Typ FILE, definice v hlavičkovém souboru stdio.h je:

```
typedef struct {
    short           level;
    unsigned        flags;
    char            fd;
    unsigned char   hold;
    short           bsize;
    unsigned char *buffer, *curp;
    unsigned        istemp;
    short           token;
} FILE;
```

# Neúplná deklarace struktur

```
struct A;                          /* neuplna */
struct B {struct A *pa};
struct A {struct B *pb};
```

Řešení je možné díky ukazateli – jeho velikost je známa, nikoli velikost struktury, na kterou ukazuje.

# Typ union

```
union [<union type name>] {
  <type> <variable names> ;
  ...
} [<union variables>] ;
```

Syntaxe jako **struct**.
Sémantika (!) – z položek unie lze používat v jednom okamžiku pouze jednu. Ostatní mají nedefinovanou hodnotu. Realizace: paměťové místo, vyhrazené pro unii je tak veliké, aby obsáhlo jedinou (paměťově největší) položku.
Je na programátorovi, pracuje–li s prvkem unie, který je určen správně či nikoliv.

# Bitová pole

Bitové pole je celé číslo, umístěné na určeném počtu bitů. Tyto bity tvoří souvislou oblast paměti. Bitové pole může obsahovat více celočíselných položek. Můžeme vytvořit bitové pole tří tříd:

1. prosté bitové pole,
2. bitové pole se znaménkem,
3. bitové pole bez znaménka.

Bitová pole můžeme deklarovat pouze jako členy *struktury* či *unie*. Výraz, který napíšeme za identifikátorem položky a dvojtečkou, představuje velikost pole v bitech.

# Struktura ftime detailně.

```
struct ftime {
    unsigned ft_tsec  : 5;  /* Two seconds */
    unsigned ft_min   : 6;  /* Minutes     */
    unsigned ft_hour  : 5;  /* Hours       */
    unsigned ft_day   : 5;  /* Days        */
    unsigned ft_month : 4;  /* Months      */
    unsigned ft_year  : 7;  /* Year - 1980 */
};
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | |
| ft_hour | | | | | ft_min | | | | | | ft_sec | | | | |
| hodiny | | | | | minuty | | | | | | sekundy/2 | | | | |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | |
| ft_year | | | | | | | ft_month | | | | ft_day | | | | |
| rok-1980 | | | | | | | měsíc | | | | den | | | | |