Třídy a struktury v C++

Petr Šaloun

katedra informatiky FEI VŠB-TU Ostrava

7. prosince 2015

Odvozené a strukturované typy dat v C

- základní datové typy – součást normy jazyka,
- preprocesor – použití netypových maker, raději volíme konstanty

```
define MAKRO hodnota
     vs
const int makro = hodnota;
```

- odvozené datové typy v C

```
typedef <type definition> <identifier>;
```

vlastní definice datového typu

- strukturované datové typy
  `struct` a `union`
  vlastní definice struktury, v C se spojuje s `typedef`.

Třídy v C++

- základ objektově orientovaného programování v C++,
- obalení, polymorfismus, dědičnost,
- tvorba tříd:
  `class`, `struct`, `union`
- v PR1 použijeme obalení (zapouzdření), pro usnadnění použijeme `struct`, konstrukci `typoedef` z C nepoužíváme.

struct ukázka

```
struct ukazka {
    int integer;
    double dbl;
    string str;
};

ukazka obj;
obj.integer = 123;
obj.dbl = 987.654;
cin >> obj.str;

cout << obj.integer << obj.dbl << obj.str;
```

Výčtový typ

umožňuje definovat konstanty výčtového typu.

```
enum [tag] {enum-list} [declarator];
```

- `enum` je kllíčové slovo, zahajuje definici hodnot výčtového typu;
- `tag` je nepovinná „visačka“, využívvaná zejména ve stylu K&R jazyka C, s konstrukcí `typedef` četnost jejího používání klesá i v C;
- `enum-list` je seznam konstant výčtového typu s možnou explicitní přiřazenou hodnotou, viz příklad dále, jinak nabývá první konstanta výčtového typu hodnoty nula, druhá hodnoty jedna, . . . , každá následující konstanta má hodnotu o jedničku vyšší;
- `declarator` je nepovinný seznam proměnných daného typu `enum`.

Použití konstant výčtového typu – 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 ...
```

Třída konstrukce struktura

V PR1 budeme používat `struct`

```
struct [<class-name>] {
  [<type> <item-name[, item-name, ...]>] ;
  [<type> <item-name[, item-name, ...]>] ;
  ...
} [<object(s)>] ;
```

. „tečka“ – selektor struktury (záznamu), např. u objektu.
-> – selektor struktury, prostřednictvím ukazatele.

Příklad struktur – struct01.cpp

```
struct complex {double re, im;};

int main() {
 complex cislo,
         im_jednotka = {0, 1};
 cislo.re = 12.3456;
 cislo.im = -987.654;
 cout << "re = " << im_jednotka.re << " im = "
      << im_jednotka.im << endl;
 cout << "re = " << cislo.re << " im = "
      << cislo.im << endl;
```

Příklad struktur – struct02.cpp

```
struct vyrobek {
  int ev_cislo;
  string nazev;
  int  na_sklade;
  double cena;
};
int main() {
 vyrobek zbozi[5];
 vyrobek a = {8765, "nazev zbozi na sklade",
              100, 123.99};
 zbozi[0].cena = 12.34;
 zbozi[0].ev_cislo = 7;
 zbozi[0].nazev = "nazev vyrobku 0";
 zbozi[0].na_sklade = 99;
 cout << a.cena << a.ev_cislo << a.nazev
      << a.na_sklade << endl;
 cout << zbozi[0].cena << zbozi[0].ev_cislo
      << zbozi[0].nazev << zbozi[0].na_sklade
      << endl;
```

Příklad struktury FILE * – datové proudy v C

Typ FILE, definice v hlavičkovém souboru cstdio je:

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

Konstrukce union v C

```
union [<union class name>] {
  <type> <item names> ;
  ...
};
```

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

Bitová 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.

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

Struktura ftime detailní ukázka.

```
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 */
 };
```