B0B99PRPA – Procedurální programování

Agregované datové typy, algoritmy řazení

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Agregované datové typy

  Struct

  Union

- Část 2 – Algoritmy řazení

  Bubble sort

  Quick Sort

Část I

Agregované datové typy

I. Agregované datové typy

Struct

Union

Struktura `struct`

- Struktura je konečná množina prvků (proměnných), které nemusí být stejného typu
- Skladba struktury je definovaná uživatelem jako nový typ sestavený z již definovaných typů
- K prvkům struktury přistupujeme **tečkovou notací**
- K prvkům můžeme přistupovat přes ukazatel operátorem `{>`
- Pro struktury stejného typu je definována operace přirazení

  `struct1 = struct2;`

  Na rozdíl od pole, kde je přiřazení nutné realizovat po prvcích.
- Struktury (jako celek) nelze porovnávat relacním operátorem `==`
- Struktura může být funkci předávána hodnotou i ukazatelem
- Struktura může být návratovou hodnotou funkce

Struktura struct – příklad

- Bez zavedení nového typu (typedef) je nutné pred identifikátor jména struktury uvádět klíčové slovo struct

```
struct record {
  int number;
  double value;
};
```

```
typedef struct {
  int n;
  double v;
} item;
```

```
record r; /* NELZE! - typ record neni znam */
struct record r; /* vyzadovano kl. slovo struct */
item i; /* definice typu pomoci typedef */
```

- Zavedením nového typu typedef můžeme používat typ struktury již bez uvádění klíčového slova struct

Definice jména struktury a typu struktury

- Uvedením `struct record` zavádíme nové jméno struktury `record`

```
struct record {
  int number;
  double value;
};
```

Definice idenfikátoru `record` ve jmeném prostoru struktur

- Definicí typu typedef zavádíme nové jméno typu record

```
typedef struct record record;
```

Definujeme globální idenfikátor record jako jméno typu struct record

- Obojí lze kombinovat v jediné definici jména a typu struktury

```
typedef struct record {
  int number;
  double value;
} record;
```

Struktura `struct` - příklad

- Struktury:

```
struct record {
  int number;
  double value;
};
```

```
typedef struct {
  int n;
  double v;
} item;
```

- Proměnné typu struktura můžeme inicializovat prvek po prvku

```
struct record r;
r.value = 21.4;
r.number = 7;
```

- Podobně jako pole lze inicializovat přímo pri definici

```
item i = { 1, 2.3 };
```

- nebo pouze konkrétní položky (ostatní jsou nulovány)

```
struct record r2 = { .value = 10.4};
```

Struktura jako parametr funkce

- Struktury můžeme předávat jako parametry funkcí hodnotou

```
void print_record(struct record rec) {
  printf("record: number(%d), value(%lf)\n",
  rec.number, rec.value);
}
```

- Nebo ukazatelem

```
void print_item(item *v) {
  printf("item: n(%d), v(%lf)\n", v->n, v->v);
}
```

- Pri predávání parametru
  - **hodnotou** se vytváří nová proměnná a původní obsah předávané struktury se kopíruje na zásobník
  - **ukazatelem** se kopíruje pouze hodnota ukazatele (adresa) a pracujeme tak s původní strukturou

Struktura `struct` – přiřazení

- Struktury:

```
struct record {
  int number;
  double value;
};
```

```
typedef struct {
  int n;
  double v;
} item;
```

```
struct record rec1 = { 10, 7.12 };
struct record rec2 = { 5, 13.1 };
item i;
print_record(rec1); /* number(10), value(7.12) */
print_record(rec2); /* number(5), value(13.10) */
rec1 = rec2;
i = rec1; /* NELZE! */
print_record(rec1); /* number(5), value(13.10) */
```

Struktura `struct` – kopie paměti

Jsou-li dve struktury stejně veliké, můžeme přímo kopírovat obsah příslušné paměťové oblasti

```
struct record r = { 7, 21.4};
item i = { 1, 2.3 };
print_record(r); /* number(7), value(21.400000) */
print_item(\&i); /* n(1), v(2.300000) */
if (sizeof(i) == sizeof(r)) {
  printf("i and r are of the same size\n");
  memcpy(\&i, \&r, sizeof(i));
  print\_item(\&i); /* n(7), v(21.400000) */
}
```

- V tomto případě je interpretace hodnot v obou strukturách identická, obecně tomu však být nemusí

Struktura `struct` a velikost

- Vnitřní reprezentace struktury nutně nemusí odpovídat součtu velikostí jednotlivých prvků

```
struct record {
  int number;
  double value;
};
```

```
typedef struct {
  int n;
  double v;
} item;
```

```
printf("int: %lu double: %lu\n", sizeof
(int), sizeof(double));
printf("record: %lu\n", sizeof(struct record));
printf("item: %lu\n", sizeof(item));
int: 4 size of double: 8
record: 16
item: 16
```

Struktura `struct` a velikost 1/2

- Při kompilaci zpravidla dochází k zarovnání prvků na velikost slova příslušné architektury

  Např. 8 bytů v případě 64 bitové architektury.

- Můžeme explicitně předepsat kompaktní pamětovou reprezentaci, např. direktivou `__attribute__((packed))` pro překladače `clang` a `gcc`

```
struct record {
  int n;
  double v;
} __attribute__((packed));
```

Struktura struct a velikost 2/2

- Nebo

```
typedef struct __attribute__((packed)) {
  int n;
  double v;
} item;

printf("int: %lu double: %lu\n", sizeof(int),
sizeof(double));
printf("record: %lu\n", sizeof(struct record));
printf("item: %lu\n", sizeof(item));
int: 4 double: 8
record: 12
item: 12
```

- Zarovnání zpravidla přináší rychlejší přístup do paměti, ale zvyšuje paměťové nároky

I. Agregované datové typy

Struct

Union

Proměnné se sdílenou pamětí – union

- Množina prvků (proměnných), které nemusí být stejného typu
- Prvky unionů sdílejí společně stejná paměťová místa

Prekrývají se.

- Velikost unionu je dána velikostí největšího z jeho prvků
- Skladba unionu je definována uživatelem jako nový typ sestavený z již definovaných typů
- K prvkům unionu se přistupuje tečkovou notací
- Pokud nedefinujeme nový typ, je nutné k identifikátoru proměnné unionu uvádět klíčové slovo union

```
union Nums {
  char c;
  int i;
};
Nums nums; /* NELZE! typ Nums neni znam! */
union Nums nums;
```

Příklad union 1/2

```
union Numbers {
  char c;
  int i;
  double d;
};

printf("char %lu\n", sizeof(char));
printf("int %lu\n", sizeof(int ));
printf("double %lu\n", sizeof(double));
printf("Numbers %lu\n", sizeof(union Numbers));

union Numbers numbers;

printf("Numbers c: %d i: %d d: %lf\n", numbers.c,
    numbers.i, numbers.d);
```

Příklad union 2/2

```
numbers.c = 'a';
printf("\nSet the numbers.c to 'a'\n");
printf("Numbers c: %d i: %d d: %lf\n", numbers.c,
    numbers.i, numbers.d);

numbers.i = 5;
printf("\nSet the numbers.i to 5\n");
printf("Numbers c: %d i: %d d: %lf\n", numbers.c,
    numbers.i, numbers.d);

numbers.d = 3.14;
printf("\nSet the numbers.d to 3.14\n");
printf("Numbers c: %d i: %d d: %lf\n", numbers.c,
    numbers.i,
numbers.d);
```

Inicializace union

- Proměnnou typu union můžeme inicializovat při definici

```
union {
  char c;
  int i;
  double d;
} numbers = { 'a' };
```

Pouze první položka (proměnná) může být inicializována.

- V C99 můžeme inicializovat konkrétní položku (proměnnou)

```
union {
  char c;
  int i;
  double d;
} numbers = { .d = 10.3 };
```

Část II

Algoritmy řazení

II. Algoritmy řazení

Bubble sort

Quick Sort

Bubble sort

- Jeden z nejjednodušších algoritmů řazení

```
void bubbleSort(int arr[], int n) {
  int i, j;
  for (i = 0; i < n-1; i++)

    // poslednich i prvku jiz serazeno
    for (j = 0; j < n-i-1; j++)
    if (arr[j] > arr[j+1])
      swap(&arr[j], &arr[j+1]);
}

void swap(int *xp, int *yp) {
  int temp = *xp;
  *xp = *yp;
  *yp = temp;
}
```

II. Algoritmy řazení

Bubble sort

Quick Sort

qsort

- Funkce v `stdlib.h`
- Implementuje Quick sort

```
int main(int argc, char *argv[]) {
  int pole[9] = {1,5,9,3,6,4,89,23,11};

  qsort((void*)pole, 9, sizeof(int), porovnejInt);

  for(int i=0; i<9; i++)
    printf("%3d, ",pole[i]);
  printf("\n");

  return 0;
}
```

qsort – porovnávací funkce

```
int porovnejInt(const void *a, const void *b) {
  int aa = *(int*)a;
  int bb = *(int*)b;
  if (aa<bb) return -1;
  if (aa>bb)return 1;
  return 0;
}
```

- parametry jsou netypované ukazatele
- vrací int
  - jestliže je první prvek menší, pak záporné číslo
  - jestliže je větší, pak kladné číslo
  - pokud jsou stejné, nulu