Štruktúry/zložené údajové typy

Náhradná Prednáška 8, PROG-2, 2020
Pavol Zajac

Zložené dátové typy: definícia typu

Hovorili sme, že v jazyku C existujú rôzne:

* základné dátové typy (`char, int, double`), z ktorých vieme odvodiť ďalšie typy pomocou modifikátorov (`long int`).
* adresy majú špecifický smerníkový typ (napr. `int*`), odvodený od typu, ktorý na danej adrese očakávame.
* polia predstavujú postupnosti prvkov rovnakého typu, pričom pri vytváraní poľa špecifikujeme, koľko prvkov sa tam objaví (`int pole[10];`)

Jazyk C umožňuje vytvárať zložené dátové typy, ktoré predstavujú logické združenie viacerých údajov (vo všeobecnosti rôzneho typu) do jedného logického celku. Vysvetlíme si na príkladoch.

Najprv si pripomenieme operátor `typedef`, ktorým môžeme premenovať typ na nami zvolený:

```
//typedef [PRESNA DEKLARACIA TYPU] [NAS NAZOV];
typedef double gps_ns;
typedef double gps_ew;

gps_ns lat;
gps_ew lon;
```

V príklade sú vytvorené dve premenné `lat` a `lon`, ktoré fyzicky v pozadí sú obyčajné `double` premenné. Logické rozdelenie typu novým pomenovaním nám však umožní napr. vedieť, že premenná `lat` je typu `gps_ns`, a teda by v nej mali byť hodnoty v intervale (-90, 90): prekladač to síce nekontroluje, ale môže to robiť programátor, alebo sa to dá testovať na úrovni QA (testovanie kódu/programov, ručné alebo automatizované). Mohli by sme tiež vymeniť podkladový typ double, napr. ak by sme chceli reprezentovať súradnice reťazcami (ale to zväčša vyžaduje aj ďalšie zásahy do kódu).

Premenné `lat` a `lon` spolu logicky súvisia: na popis nejakého miesta na Zemi potrebujeme obidve súradnice. Predtým ako budete študovať ďalej, skúste si naprogramovať iba s doterajším arzenálom jednoducho znejúcu úlohu: *Funkcia dostane pole miest na Zemi (daných gps súradnicami) a jedno extra miesto, usporiadajte pole súradníc podľa vzdialenosti od zadaného bodu.*

***TODO**: naprogramujte si zadanú úlohu, na výpočet vzdialenosti môžete použiť hotovú funkciu z* https://uim.fei.stuba.sk/b-prog2/b-prog2-materialy-2020/*, druhá prednáška, gps.c*

Jazyk C nám umožňuje združiť aj fyzicky tieto dve premenné do jedného celku, GPS súradnice, pomocou zloženého dátového typu: štruktúry. Zapíšeme to nasledovne:

```
struct { gps_ns lat; gps_ew lon; }; // ; na zaver je nutna...
```

***TODO**: naštudujte si presnú syntax definície štruktúry z literatúry/online zdrojov*

Uvedená syntax nám zadefinovala nepomenovanú štruktúru. Na každom mieste, kde by sme tento typ chceli použiť (vytváranie premenných, funkcií, polí, smerníkov), museli by sme použiť plný zápis. Aby sme sa tomu vyhli, môžeme:
a) použiť pomenovanú štruktúru:

```
// _gps je nas nazov struktury, tzv. tag
struct _gps { gps_ns lat; gps_ew lon; };
struct _gps gps;   //premenna typu struct _gps
```

Pozor, na rozdiel od C++, v jazyku C sa musí používať celý typ, t.j. `struct _gps`*, nie iba* `_gps`.

b) využiť `typedef`:

```
typedef struct _gps { gps_ns lat; gps_ew lon; } GPS;
GPS gps;   //premenna vlastneho typu GPS, iny nazoc struct _gps
```

Pozn.: Keď používame `typedef`*, tag nepotrebujeme. Ak ho použijeme, môžeme použiť oba typy (v príklade aj* `struct _gps`*, aj* `GPS`*. Pri používaní vlastného typu* `GPS` *už keyword* `struct` *nepoužívame .*

V definícii štruktúry vnútorné prvky označujeme ako položky (niekde sa používa aj "fields", ale v slovenčine je preklad polia, a to by bolo mätúce). Ako položky je možné používať akékoľvek deklarácie, ako keby sme vytvárali lokálne premenné ľubovoľného známeho typu, vrátane polí, smerníkov a iných štruktúr. Napr. môžeme definovať lokalitu jej názvom /reťazec do 30 znakov dĺžky/ a zemepisnou súradnicou:

```
typedef struct _lokalita {
   char nazov[31];
   GPS gps;
} Lokalita;
```

Pozn.: Všimnite si, že som si zvolil pomenovaciu konvenciu: `struct` *tag začínam znakom _, a názov* `typedef`*-ového typu veľkým písmenom, premenné malým písmenom. Keďže C je case sensitive, umožňuje mi to písať dočasné premenné štýlom*

```
  Lokalita lokalita;
```

Pomenovacia konvencia nie je niečo, čo je predpísané syntaxou jazyka C (a teda povinné), ale je to niečo navyše, načom sa programátori dohodnú, aby sa im lepšie čítal /a písal, a opravoval/ kód.

Pozn.: Obsah štruktúr by mal odrážať nejaké reálne údaje, ktoré modelujete. Ale programátor si môže definovať štruktúry ako potrebuje. Napr. v zadaní 3 môžete nastavenia spracované z command line cez getopt ukladať do štruktúry, kde ich budete mať všetky pokope, a nie v X premenných.

Zložené dátové typy: štruktúrové premenné, inicializácia

Keď už máme definované vhodné typy (definície typov sa často dávajú do samostatných vlastných hlavičkových súborov, to by ste si mali odskúšať v zadaní 4 a 5), môžeme používať tieto typy podobne ako základné typy: dajú sa vytvárať štruktúrové premenné, funkcie môžu mať štruktúrové parametre a môžu vracať štruktúru (štruktúry sa odovzdávajú hodnotou, teda sa kopírujú, viac neskôr), môžeme vytvoriť polia štruktúr, smerníky na štruktúrové údaje, ...

```
GPS gps1, gps2, *p;  //dve GPS suradnice, a jeden smernik na GPS
Lokalita lokalita;   //jedna premenná pre lokalitu
Lokalita data[100];  //pole 100 lokalit
```

Premenné štruktúrového typu inicializujeme pri deklarácii podobne ako polia, do {} dáme v poradí obsahy príslušných zložiek štruktúry. *POZOR: Ak sa inicializácia pri lokálnej premennej neuvedie, jej obsah bude nedefinovaný (pri globálnych a statických sa inicializujú na 0).*

```
GPS gps1 = {48.14816, 17.10674}; //Bratislava
Lokalita fei = {"FEI STU", {48.1518532,17.0711559}};
GPS pole[10] = {{48.14816, 17.10674}, {48.1518532,17.0711559}};
```

Všimnite si: Pri deklarácii `fei` sme použili syntax naplnenia poľa znakov z reťazcovej konštanty, a položku `gps` sme nastavili pomocou syntaxe pre štruktúru, teda sme potrebovali vnorené `{}`. Podobne v poli, vonkajšie `{}` sú pre pole, vnútorné pre jednotlivé GPS prvky poľa (2 sú nastavené, zvyšné sú nulové).

V pamäti funkcie sa vytvorí zhruba niečo takéto (*píšem zhruba, lebo presný pamäťový obraz závisí od nastavenia prekladača, na rozdiel od poľa, položky štruktúry nemusia byť v pamäti súvisle za sebou*):

0x80100	48.14816 17.10674	gps1
0x80120	F E I S T U \0 48.1518532 17.0711559	fei
0x80200	48.14816 17.10674	pole[0]
0x80220	48.1518532 17.0711559	pole[1]
0x80240	0.0 0.0	pole[2]
...	...	...

*Pozn.: V štandarde C99 existuje **designated initializer**, kde môžete explicitne nastaviť iba niektoré položky, alebo ich nastaviť v ľubovoľnom poradí. Vyhľadajte si, ako sa to robí.*

Štruktúrové premenné, operátory

Na rozdiel od bežných číselných premenných, so štruktúrovými premennými sa nedá robiť ľubovoľné operácie. Nie sú definované operátory aritmetické, logické (napr. porovnanie), nevieme použiť printf s nejakým jednoduchým formátovacím reťazcom. Všetky operácie so štruktúrami si musíme naprogramovať vo forme vhodných funkcií, alebo používať priamo položky štruktúr. Jazyk C nám poskytuje možnosť:

* kopírovať obsah štruktúry (presný pamäťový obraz), pomocou operátora =, ako vstup do funkcie, alebo cez `return` ako výsledok funkcie,
* sprístupňovať položky štruktúry pomocou operátora `.`

Príklad:

```
GPS gps1 = {48.14816, 17.10674}; //Bratislava
Lokalita ba;                     //nenastavena, nastavime rucne

ba.gps = gps1;
//ba.gps spristupni polozku gps z premennej ba
// operator = skopiruje cely obsah premennej gps1 do ba.gps

strcpy(ba.nazov, "Bratislava");
//ba.nazov je standardne pole znakov, cize s nim pracujeme,
// ako s beznym miestom na retazec,
// vratane toho, ze moze dojst k preteceniu, ak si nedame pozor

printf("%s ma suradnice %.5lf %.5lf\n",
          ba.nazov,
//spristupnime adresu zaciatku retazca nazov v ramci lokality ba
          ba.gps.lat,
//spristupnime najprv gps suradnicu ba, a z nej udaj lat
          ba.gps.lon);
//a potom aj lon
```

Príklad s funkciami. Štruktúry nám zjednodušujú zápis parametrov a sprehľadňujú kód:

```
//vstupom je lokalita, formatovane ju vypise
void vypis_lokalitu(Lokalita lokalita) {
     printf("%s ma suradnice %.5lf %.5lf\n",
          lokalita.nazov,
          lokalita.gps.lat,
          lokalita.gps.lon);
}
int main() {
     Lokalita ba, fei;
     //TODO: podobny kod ako v priklade vyssie na nastavenie ba a fei
     vypis_lokalitu(ba);   //cela ba sa nakopiruje do lokalita
     vypis_lokalitu(fei);  //a potom sa to zopakuje s fei
}
```

Na rozdiel od polí, funkcie môžu vrátiť celé štruktúry. Vďaka tomu vieme napr. z funkcií vrátiť aj viac údajov spolu (ako jediný štruktúrovaný údaj) aj bez použitia smerníkov. Ak funkcia vracia štruktúru, zväčša potrebujeme vo volajúcej funkcii mať príslušnú premennú, do ktorej sa obsah vrátenej štruktúry nakopíruje.

Príklad:

```
GPS random_gps() {
     //docasna premenna, zije v tejto funkcii iba po return
     GPS gps;

     //vypocty robime priamo na zlozkach docasnej premennej
     gps.lat = -90.0 + 180.0 * rand()/(double)RAND_MAX;
     gps.lon = -180.0 + 360.0 * rand()/(double)RAND_MAX;

     //tu nastane magia: prekladac to prepoji s volajucou funkciou
     // a nakopiruje docasne gps kam treba
     return gps;
}
int main() {
     Lokalita test;
     strcpy(test.nazov, "Random");
          //naplni sa gps suradnica podla vysledku funkcie
     test.gps = random_gps();
     vypis_lokalitu(test);
}
```

Smerníky a štruktúry

Pri použití štruktúr ako parametrov funkcií a návratových hodnôt z funkcií sa obsah štruktúry kopíruje v pamäti. Štruktúry môžu byť niekedy dosť veľké (do jedného štruktúrovaného údaja môžete zabaliť ďalšie podštruktúry a celé polia), preto používanie štruktúr ako parametrov môže značne spomaliť volanie funkcií. Pri použití štruktúr ako parametra vo volanej funkcii pracujete s kópiou dát, štruktúra v pôvodnej funkcii sa nezmení (podobne, ako keby ste posielali napr. `int`). To môže byť výhoda aj nevýhoda v závislosti na účele funkcie, ktorú vytvárate.

Príklad na **nesprávne** použitie štruktúry:

```
//chceme nacitat gps suradnicu z klavesnice cez scanf
void nacitaj_gps(GPS gps) {
     printf("Zadaj zemepisnu sirku: ");
     scanf("%lf", &gps.lat);
     //pozn.: operator . ma vyssiu prioritu
     // najprv sa spristupni gps.lat
     // a potom pomocou & sa adresa tohto double posle do scanf

     printf("Zadaj zemepisnu dlzku: ");
     scanf("%lf", &gps.lon);
}
```

Vyskúšajte si príklad otestovať, napr.:

```
int main() {
     Lokalita vlastna = {"test", {0.0,0.0}};
     //pokusme sa nacitat obsah vlastna.gps z klavesnice
     nacitaj_gps(vlastna.gps);
     vypis_lokalitu(vlastna);
}
```

Ak si spustíte príklad, program sa spýta na súradnice, ale vypíše 0 0 bez ohľadu na to, čo zadáte. Ak si príklad odkrokujete, uvidíte, že vo funkcii `nacitaj_gps` sa premenná `gps` zmení. Toto je však nová lokálna premenná /nastavená na začiatku na kópiu hodnôt, ktoré boli vo `vlastna.gps`/. Po skončení funkcie zmeny zmiznú spolu s touto dočasnou premennou.

Jedno z riešení je vrátiť nový obsah a použiť návratovú funkciu vo volajúcej funkcii. Príklad:

```
//chceme nacitat gps suradnicu z klavesnice cez scanf
GPS nacitaj_gps2() {
     GPS gps;
//teraz to nie je parameter, kedze ju nechceme nastavit pri volani

     printf("Zadaj zemepisnu sirku: ");
     scanf("%lf", &gps.lat);
     //pozn.: operator . ma vyssiu prioritu
     // najprv sa spristupni gps.lat
     // a potom pomocou & sa adresa tohto double posle do scanf

     printf("Zadaj zemepisnu dlzku: ");
     scanf("%lf", &gps.lon);

     //vratime naplnenu gps suradnicu volajucej funkcii
     return gps;
}

int main() {
     Lokalita vlastna = {"test", {0.0,0.0}};
     //nacitame gps suradnicu z klavesnice cez funkciu
     // a skopirujeme vysledok do vlastna.gps
     vlastna.gps = nacitaj_gps2();
     vypis_lokalitu(vlastna);
}
```

Druhým riešením je použiť nepriamy prístup k štruktúre pomocou smerníkov (na štruktúru). Smerníky na štruktúry fungujú rovnako ako iné smerníky: sú to len premenné, ktorých obsahom je adresa zvoleného štruktúrovaného údaju.

```
GPS gps = { 48.14816, 17.10674 };
GPS *ptr = &gps;
```

Do premennej `ptr` sme uložili adresu premennej `gps`. So smerníkom ptr pracujeme štandardne ako s ľubovoľným iným smerníkom: dá sa posúvať (smerníková aritmetika), dá sa sprístupniť (operátor `*`, alebo operátor `[]`), ... Pri sprístupnení cez smerník, výsledkom je celá štruktúra zvoleného typu.

```
Lokalita vlastna = {"test", {0.0,0.0}};
GPS gps = { 48.14816, 17.10674 };
GPS *ptr = &gps;

vlastna.gps = *ptr;
```

Smerníky môžu ukazovať aj do vnútra štruktúry. V tom prípade typ smerníku by mal zodpovedať typu položky. Príklady:

```
Lokalita vlastna = {"test", {0.0,0.0}};
GPS gps = { 48.14816, 17.10674 };
GPS *ptr;

ptr = &vlastna.gps;      //operator . ma prednost
*ptr = gps;              //zmenime obsah iba casti lokality

double* p = &vlastna.gps.lat;
scanf("%lf", p);
//POZOR: tu v scanf uz nedavame &p, lebo zelana adresa je obsahom p,
// nie adresou p
```

Pozn.: aj keď v definícii štruktúry nasleduje za `lat` položka `lon`, nie je garantované, že `p+1` bude ukazovať na `vlastna.gps.lon`. Prekladač môže optimalizovať rozloženie položiek v štruktúre, môže medzi nimi byť priestor, alebo byť inak usporiadané. Smerníková aritmetika správne funguje len s poľami /vrátane prípadov, keď je pole vnútri štruktúry/.

Keď máme smerník na štruktúru, prostredníctvom operátora `*` sa vieme dostať k obsahu štruktúry ako celku. Keď chceme jednotlivé položky štruktúry, môžeme skombinať operátory `*` a `.`. Problém je v priorite operátorov, C sa snaží najprv vyhodnotiť sprístupnenie položky a potom `*`.

Príklad:

```
Lokalita ba = {"Bratislava",{ 48.14816, 17.10674 }};
Lokalite *ptr = &ba;
```

`*ptr.gps.lat` bude nedefinované, lebo `ptr` je smerník, a nepodporuje operátor .
`(*ptr).gps.lat` korektne sprístupní obsah, kam ukazuje `ptr`, stade prečíta položku `.gps.lat`

Na zjednodušenie syntaxe `(*ptr).polozka` je v jazyku C definované sprístupnenie položky štruktúry cez adresu štruktúry. Slúži na to operátor `->` (to je o 2 znaky menej, ako zátvorkovaná * a ., plus je to čitateľnejšie, keď si zvyknete na smerníky na štruktúry). Operátor `->` môžeme použiť len s adresou vľavo, nie so štruktúrou ako takou.

`ptr->gps.lat` sprístupní to isté ako zápis `(*ptr).gps.lat`

`(*ptr).nazov` bude typu `char*`, a bude odkazovať na reťazec `"Bratislava"`
`ptr->nazov` je to isté, čo o riadok vyššie, ale zapísané smerníkovou syntaxou

`*ptr.nazov` nie je definované, lebo `ptr` je smerník, nie štruktúra
`(*ptr)->nazov` nie je definované, lebo `*ptr` nie je smerník, ale štruktúra

`*ptr->nazov` je korektný zápis a bude mať hodnotu `'B'` typu `char`: najprv sa zistí adresa štruktúry cez `ptr`, z nej sa prečíta hodnota `ptr->nazov`, čo je adresa typu `char*`, potom sa aplikuje `*` a vyberie sa z pamäti znak na začiatku `ptr.nazov`.

Smerníky na štruktúry majú značné využitie: Keď máme štruktúry s veľa dátami, je možné do funkcie poslať iba smerník na štruktúru, a dáta sa nebudú kopírovať /podobne ako pri poliach/. Hovoríme o odovzdávaní údajov odkazom. Ak funkcii dáme smerník na štruktúru, funkcia bude môcť meniť dáta na pôvodnom mieste. Ak to nie je želané, treba smerník kvalifikovať ako `const`. Neskôr si ukážeme ďalšie zložitejšie použitia smerníkov na štruktúry pri riešení rôznych úloh.

Príklady na jednoduché funkcie so smerníkmi na štruktúry:

```
//vstupom je lokalita odovzdana odkazom, kvoli rychlosti,
// kedze lokalitu len citame, pouzijeme const
void vypis_lokalitu_ptr(const Lokalita *lokalita) {
     printf("%s ma suradnice %.5lf %.5lf\n",
          lokalita->nazov,
          lokalita->gps.lat,
          lokalita->gps.lon);
          //v tejto funkcii lokalita je smernik a vrati adresu
          // na spristupnenie poloziek pouzijeme ->
          // lokalita->gps nie je adresa, ale struktura
          // na spristupnenie poloziek gps pouzijeme .
}
int main() {
     Lokalita ba = {"Bratislava",{ 48.14816, 17.10674 }};;

     vypis_lokalitu_ptr(&ba);
     //vstupny parameter je teraz adresa, nie kopia struktury
}
```

Príklad na načítanie z klávesnice pomocou adresy štruktúry:

```
//chceme nacitat gps suradnicu z klavesnice cez scanf
// vstupom je adresa struktury, kam sa suradnica ulozi
int nacitaj_gps3(GPS *gps) {

     printf("Zadaj zemepisnu sirku: ");
     scanf("%lf", &gps->lat);
     //pozn.: operator -> ma vyssiu prioritu
     // najprv sa spristupni gps->lat
     // a potom pomocou & sa adresa tohto double posle do scanf
```

```
    printf("Zadaj zemepisnu dlzku: ");
    scanf("%lf", &gps->lon);

    //vratime nejaky ciselny kod,
    // vo funkcii by sme mohli napr. osetrit, ci scanf boli ok
    return 1;
}

int main() {
    Lokalita vlastna = {"test", {0.0,0.0}};
    //do funkcie posielame adresu nejakej GPS struktury
    // . ma prednost, najprv sa spristupni vlastna.gps
    // a cez & zisti jej adresa, ktora sa posle do funkcie
    if (nacitaj_gps3(&vlastna.gps));
        vypis_lokalitu(vlastna);
}
```

TODO: K prednáške prikladám zdrojové príklady preberaných funkcií bez smerníkov aj s nimi, vyskúšajte si ich.

TODO: Vyskúšajte si naprogramovať na začiatku uvedenú úlohu tentoraz s pomocou štruktúr, ktoré boli použité v prednáške. Znenie úlohy: *Funkcia dostane pole miest na Zemi (zadaných gps súradnicami) a jedno extra miesto, usporiadajte pole súradníc podľa vzdialenosti od zadaného bodu.*