45. Programátorské paradigmy XIV.

Polia a ich indexy

```
{
  uint8_t i;
  char buf[20], s[20];

  ioinit();
```

Aj keď polia premenných predstavujú základnú dátovú štruktúru a sú prítomné snáď v každom programovacom jazyku, v jazyku C sú polia implementované v charakteristicky zjednodušenej podobe. S určitou nadsádzkou by sa takmer dalo povedať aj to, že v jazyku C polia ani neexistujú. Presnejšie povedané, existuje spôsob ako definovať pole premenných a v prípade potreby ho inicializovať, následne je však pri použití **meno poľa konvertované na smerník** na typ daný typom premennej poľa[1]. Prístup k jednotlivým prvkom poľa sa deje s využitím smerníkovej aritmetiky, i keď prostredníctvom zápisu formálne pripomínajúcom indexovanie prvkov poľa.

Deklarácia premennej `char buf[20]` teda vyhradí v dátovej pamäti priestor pre 20 premenných typu `char`, a vytvorí symbol (konštantu) `buf`, ktorý je pri použití konvertovaný na typ `char*` s hodnotou zhodnou so smerníkom na prvý prvok poľa, t.j. `&buf[0]`. **Výraz `buf[i]`** v jazyku C **je plne ekvivalentný výrazu `*(buf + i)`**[2], pričom si treba uvedomiť, že vďaka zásadám smerníkovej aritmetiky nejde o „obyčajné“ sčítanie, ale sa takto k hodnote smerníka `buf` pripočíta `i`-násobok veľkosti pamäte, ktorú základný typ smerníka – a tu aj poľa – zaberá. Tento prístup je aj dôvodom toho, že v jazyku C sa nedá voliť rozsah indexov poľa, **indexy sú pevne od 0 do N-1** (kde N je deklarovaný počet prvkov poľa).

Jednotlivé prvky poľa sú umiestnené v pamäti tesne jedna za druhou, bez akéhokoľvek zarovnania. Táto požiadavka zaručuje, že sa dá použiť smerníková aritmetika na pristupovanie k prvkom poľa. Následkom tohto je aj fakt, že prvý byte i-teho prvku poľa sa nachádza na adrese, ktorá je oproti adrese začiatku poľa posunutá o `i * sizeof(buf)`.

Zjednodušený prístup k poliam prostredníctvom smerníkovej aritmetiky má za následok aj to, že je možné vo výraze typu `buf[i]` použiť akýkoľvek celočíselný index, vrátane hodnôt väčších ako je deklarovaný počet prvkov poľa, ba dokonca aj záporné hodnoty; samozrejme, s použitím indexov mimo deklarovaného rozsahu poľa budú pristupované iné premenné, než jednotlivé prvky poľa. A nielen to - ani symbol pred indexom nemusí vôbec byť menom deklarovaného poľa, môže to byť akýkoľvek smerník či smerníková konštanta.

Automatickú kontrolu hraníc indexov poľa definícia jazyka teda nevyžaduje, a to ani nie tak z nejakých principiálnych dôvodov, ako skôr kvôli (mimochodom ani nie príliš významnému) zjednodušeniu prvého prekladača jazyka C. Keďže ten bol vzorom pre prvú verziu defacto štandardu, táto zásada v jazyku už zostala pevne ukotvená... Očakáva sa, že kontrolu hraníc indexov bude písať explicitne programátor, ktorý môže odhadnúť, kde je skutočne nevyhnutná a tak bude program pracovať optimálnejšie než pri automatickej kontrole. Ako však už prax neraz ukázala, programátori majú tendenciu takýto druh „nadpráce“ ignorovať. Toto vedie nielen k potenciálnym chybám, ale ukázalo sa, že z tohto vyplývajúci tzv. *buffer overflow attack* je aj jeden z najvýznamnejších bezpečnostných problémov mnohých, najmä webových, aplikácií.

1 C99, §6.3.2.1, ods.3
2 C99, §6.5.2.1; kurióznym dôsledkom tohto pravidla je, že tento výraz je možné zapísať aj ako `i[buf]`

To, že prekladač nerobí kontrolu hraníc poľa, však neznamená, že je možné tento fakt "beztrestne" a bezmedzne využívať. Prekladač, ale aj hardware môže v niektorých prípadoch kontrolu hraníc poľa či nejakého väčšieho úseku pamäte implementovať. Štandard dokonca priamo upozorňuje, že pokus o operáciu s využitím smerníkovej aritmetiky (čo je aj použitie idexu poľa v []), ktorého výsledkom je smerník, ktorý neukazuje na prvok objektu na ktorý ukazoval pôvodný smerník, má nedefinovaný výsledok (čo treba chápať tak, že môže skončiť chybou počas behu, alebo aj počas prekladu). Ako pomerne kuriózna výnimka, štandard dovoľuje, aby výsledkom bol smerník ktorý ukazuje presne jednu položku za objekt, avšak takýto výsledok (pochopiteľne) nesmie byť dereferencovaný.

Pri deklarácii poľa nie je vyžadované uviesť explicitne počet jeho prvkov, ak je pole inicializované. V takom prípade zostanú hranaté zátvorky za menom poľa prázdne a počet prvkov poľa je daná inicializátorom. V inicializátore poľa (či už má explicitne uvedený počet prvkov alebo nie) je možné predpísať, ku ktorému prvku poľa element inicializátora patrí; elementy inicializátora nasledujúce za takýmto elementom, ktoré nemajú predpísanú príslušnosť k prvku poľa, sú ukladané do postupne nasledujúcich prvkov. Prvky poľa, ktoré nemajú v inicializátore predpísaný obsah, sú inicializované na 0 (toto samozrejme platí len pre polia ktoré sú deklarované s inicializátorom; pre polia bez inicializátora platia všeobecné pravidlá pre premenné, t.j. globálne a statické lokálne premenné sú inicializované na 0; lokálne nestatické premenné nie sú inicializované a môžu mať pri každom začatí vykonávania bloku, v ktorom sú deklarované, náhodný obsah). Napríklad, pole deklarované nasledujúcim spôsobom:

```
int a[5] = {[3] = 1, [0] = 2, 3};
```

má jednotlivé prvky inicializované na hodnoty 2, 3, 0, 1, 0.

V jazyku C je možné deklarovať aj viacrozmerné polia, jednoducho zoradením viacerých hranatých zátvoriek za menom poľa. Takéto pole sa interpretuje v smere od prvej hranatej zátvorky k poslednej, takže napríklad deklarácia `char buf[5][2]` je 5-prvkovým poľom, ktorého prvky sú 2-prvkové polia prvkov typu `char`. Inakšie povedané, pri pohybe po poli v pamäti sa najrýchlejšie mení posledný index. Pri deklarácii viacrozmerného poľa musí byť explicitne uvedený počet prvkov vo všetých zátvorkách s výnimkou tej, ktorá je najbližšie k menu poľa.

Niekedy sa zjednodušene uvádza, že symbol (meno) označujúci pole je smerníkom. Nie je to však správne - smerník je predsa tiež premenná a sama zaberá pamäť; nie je dokonca úplne správne tvrdiť ani to, že je to smerníková konštanta. Existujú totiž dva prípady, keď sa meno poľa takto nespráva. Prvý takýto prípad je, keď je meno poľa operandom unárneho operátora &, v tomto prípade meno označuje pole ako objekt a operátor & vracia jeho adresu v pamäti, t.j. výrazy `buf`, `&buf` a `&buf[0]` sú navzájom ekvivalentné.

Druhou výnimkou je použitie mena poľa ako argument operátora `sizeof` (pozri kapitolu 51) - ak by sa meno poľa správalo ako smerníková konštanta, operátor `sizeof` by vracal veľkosť smerníka na typ prvku poľa, štandard však určuje, že v tomto prípade operátor `sizeof` vracia celkovú veľkosť poľa, t.j. veľkosť pamäte, ktorú pole zaberá. Ako následok je možné zistiť počet prvkov poľa výrazom `sizeof(buf)/sizeof(buf[0])`, čo je praktické najmä ak pri deklarácii poľa nebol explicitne uvedený počet jeho prvkov a veľkosť poľa bola určená inicializátorom.

Táto výnimka však má tiež výnimku: ak je meno poľa použité v deklarácii parametrov funkcie, operátor `sizeof` vracia veľkosť smerníka na typ prvku poľa. Nasledujúci krátky program:

```
#include <stdio.h>
char a[3];

void foo(char b[3]) {
  printf("%d %d", sizeof(a), sizeof(b))
}

int main(void) {
  foo(a);
}
```

teda po preložení a spustení na systéme, kde veľkosť smerníka sú 4 byte (napr. bežné PC), vypíše "3 4".