36. Programátorské paradigmy VIII.

void

```
static void
delay_1s(void)
{
  uint8_t i;
```

Kľúčové slovo **void** má v jazyku C zvláštne postavenie. Aj keď sa používa na takých miestach, kde sa normálne používajú mená typov, nejedná sa o plnohodnotný typ - v terminológii normy sa jedná o **nekompletný typ** a napríklad nie je možné deklarovať premennú takéhoto typu. Pôvodný „prirodzený“ význam slova *void* je *prázdnota, prázdne miesto*, a v jazyku C je používané v niekoľkých konkrétnych situáciách ako náhradné kľúčové slovo pre nedefinovateľný typ.

Jedno z použití kľúčového slova `void` je aj v hlavičke funkcie v našom príklade, a to hneď na dvoch miestach. V pvom prípade sa ním vyznačuje formálny návratový typ funkcií, ktoré nemajú návratovú hodnotu. V tele takýchto funkcií sa nevyskytuje kľúčové slovo `return` nasledované výrazom. Kľúčovým slovom `void` sa tiež nahrádzajú formálne parametre v prípade, že funkcia žiadne parametre nemá, čo je druhý prípad použitia `void` v našom príklade. Takto je zachovaný formálny vzhľad hlavičky funkcie (t.j. štruktúra `typ - meno funkcie - parametre v zátvorke`), čím sa pre prekladač zjednocuje vyhodnotenie hlavičky funkcií, nech majú plnohodnotný návratový typ, parametre, či oboje.

Druhé významné použitie kľúčového slova `void` je pre vytvorenie nešpecifického typu smerníka `void *`. Tento typ je používaný ako „medzityp“ na formálne správne pretypovávanie smerníkov na rôzne typy - presnejšie povedané, potláča sa tým hlásenie prekladača o nekompatibilite smerníkových typov. Takéto pretypovanie slúži na pristupovanie k tým istým údajom interpretovaným iným spôsobom. Takáto „reinterpretácia“ údajov sa nazýva *type punning*[1]. Napríklad, majme pole bytov, ktoré používame ako buffer pre komunikačné rozhranie a vysielame

1 *pun* v angličtine označuje slovnú hračku založenú na podobne znejúcich slovách (homonymách), alebo na rôznych významoch toho istého slova

z neho po byte, ale chceme do neho napríklad od pozície 4 uložiť naraz celé 32-bitové slovo:

```
uint8_t buf[100];
uint32_t someWord;
*(uint32_t *)(void *)&buf[4] = someWord;
TransmitBuffer(buf);
```

Inou formou *type punning*u je za pomoci (či skôr zneužitím) `union`:

```
union {
  uint8_t b[100];
  struct {
    uint32_t size;
    uint32_t firstWord;
    [...]   // ostatné položky paketu typu A
  } packetTypeA;
} buf;
buf.firstWord = someword;
TransmitBuffer(buf.b);
```

Obe formy sú rovnako principiálne nesprávne, a pre oba prípady norma varuje, že výsledok nie je zaručený (správne sa má konverzia zapísať explicitne, napríklad viacbytové slovo rozdeliť na jednotlive byty pomocou operácií celočíselného delenia a zvyšku). V jazyku C sa však jedná o veľmi rozšírenú prax, ktorá sa prejavuje napríklad aj tým, že mnohé prekladače nenamietajú pri pretypovaní smerníkov aj bez „medzikroku“ pretypovania na `void *`. *Type punning* na jednej strane zjednodušuje pristupovanie k údajom interpretovaným rozdielnymi typmi (napr. ako pole byte-ov počas komunikácie ale ako určitá štruktúra pri ich použití), na druhej strane však spôsobuje aj problémy s prenositeľnosťou programov a komplikuje aj úlohu optimalizátora prekladača.

Vráťme sa však k typu `void`: posledné, málo známe a aj zriedkavé použitie `void` je ako cieľový typ pre pretypovanie nejakého výrazu, čoho následkom je výsledok výrazu ignorovaný. Slúži na formálne správne zapísanie samostatného výrazu, ktorého výsledok nie je dôležitý, avšak jeho vykonanie má dôležité vedľajšie efekty (napríklad čítanie z hardwarového registra procesora). Pretypovanie na `void` je tu znova nápovedou pre kompilátor, že pre takýto zápis nie je potrebné vypísať upozornenie (*warning*).

Tento istý mechanizmus sa dá použiť aj na potlačenie *warning*u kompilátora pre parametre funkcií a premenné, ktoré nikde nie sú použité. Takýto stav sa môže vyskytnúť napríklad v prípade, keď sa

premenná či parameter používa len takej časti programu, ktorá sa pri určitých nastaveniach makier preprocesora vďaka prostriedkom podmieneného prekladu neprekladá:

```
void PrintError(char * a) {
  #ifdef RELEASE
    (void)a;
  #else
    printf(a);
  #endif
}
```

V tomto príklade by síce bolo možné podmienene prekladať aj definíciu či deklaráciu premennej alebo parametra, to by však viedlo k menej čitateľnému programu najmä v prípade parametrov funkcie, kde by sa premenlivý počet parametrov musel brať do úvahy aj pri volaniach funkcie.