JAZYK C (2.)

Céčko na přelomu tisíciletí

V další části povídání o novinkách, které přinesla druhá verze standardu ISO jazyka C (ISO 9899:1999), nejprve dokončíme kapitolu o celočíselných typech a pak se podíváme na typy pro práci s reálnými a komplexními čísly a na pravidla pro konverze.

HODNOSTI

Celočíselným typům jsou nyní v jazyce C přiřazeny tzv. **konverzní hodnosti** *(conversion rank)*, které se kromě určování pravidel pro konverze uplatňují také při specifikaci rozsahu. Hodnost musí být pro různé celočíselné typy se znaménkem různá, a to i v případě, že mají stejný rozsah hodnot. Uspořádáme-li základní znaménkové typy podle hodnosti od nejvyšší, dostaneme posloupnost `long long`, `long`, `int`, `short`, `signed char`. Hodnosti typů bez znamének jsou stejné jako hodnosti odpovídajících znaménkových typů.

Nejnižší hodnost má typ `_Bool`. Typ `char` má stejnou hodnost jako typy `unsigned char` a `signed char`. Výčtové typy mají stejnou hodnost jako jejich podkladové celočíselné typy.

Pokud daná implementace obsahuje doplňkové celočíselné typy *(extended integer types)*, musí mít doplňkový typ nižší hodnost než standardní celočíselný typ se stejným rozsahem.

Množina hodnot znaménkového typu s nižší hodností musí být podmnožinou množiny hodnot znaménkového typu s vyšší hodností. Odtud plyne, že např. rozsah hodnot typu `short` nesmí být větší než rozsah hodnot typu `int`, a další podobná pravidla pro typy se znaménkem i bez něj. (Tyto zásady platily už v předchozí verzi standardu, nyní ovšem plynou z pravidla o hodnosti.)

CELOČÍSELNÁ ROZŠÍŘENÍ

Jako **celočíselná rozšíření** *(integer promotions)* se označují konverze, které se provádějí s operandy některých unárních nebo binárních operátorů. Na místě, kde se očekává celé číslo typu `int` nebo `unsigned int`, lze použít jakoukoli hodnotu typu, který má menší konverzní hodnost než tyto dva typy. Lze tam také použít hodnotu bitového pole typu `_Bool`, `signed int`, `int` nebo `unsigned int`. Pravidlo pro celočíselná rozšíření je nyní formulováno takto:

Může-li typ `int` *reprezentovat všechny hodnoty typu konvertované hodnoty, převede se tato hodnota na typ* `int`*, jinak se převede na typ* `unsigned`.

Výsledek pro základní typy je stejný jako v předchozí verzi, ovšem formulace založená na konverzních hodnostech bere v úvahu také možnost, že daná implementace bude obsahovat i další („rozšířené", doplňkové) typy.

REÁLNÁ ČÍSLA

Stejně jako stará verze standardu nám i nová dává k dispozici tři datové typy pro práci s reálnými čísly, a to `float`, `double` a `long double`. Nový standard už neuvádí `long float` jako synonymum pro `double`.

V hlavičkovém souboru `math.h` jsou definovány typy `float_t` a `double_t`, což jsou reálné typy s alespoň takovým rozsahem jako `float`, resp. `double`. Jejich skutečný význam se řídí hodnotou makra `FLT_EVAL_METHOD`: Má-li toto makro hodnotu 0, představují typy `float_t`, resp. `double_t` ve skutečnosti typy `float`, resp. `double`. Má-li hodnotu 1, představují oba `double`, a má-li hodnotu 2, představují oba `long double`. Konkrétní implementace může definovat ještě další možnosti.

REÁLNÉ LITERÁLY

Reálné literály lze nyní zapisovat také v šestnáctkové soustavě. Takovýto literál začíná prefixem `0x` nebo `0X`, za nímž následuje šestnáctkové číslo s řádovou čárkou a „exponentová část". V ní ovšem místo `e` nebo `E` musíme použít `p` nebo `P`. Tento exponent se vztahuje **k základu 2**, nikoli k základu 10 nebo 16 (!).

KONTRAHOVANÉ VÝRAZY

Reálné výrazy mohou být **kontrahované** *(contracted)*. To znamená, že se vyhodnocují jako atomické operace a přitom se ignorují zaokrouhlovací chyby dané způsobem vyhodnocení. Nemusí dokonce dojít ani k vyvolání výjimek typických pro reálné operace, které by jinak vznikly.

K povolení, resp. zakázání kontrahovaných výrazů slouží standardní direktiva

```
#pragma STDC FP_CONTRACT ON
```

resp.

```
#pragma STDC FP_CONTRACT OFF
```

ZVLÁŠTNÍ HODNOTY

Nová verze standardu podporuje mj. počítání se „zvláštními" hodnotami reálných čísel, pokud jsou v dané architektuře k dispozici.

Makro `INFINITY` (opět definované v `math.h`) se rozvine v kladné nekonečno nebo v nekonečno bez znaménka, pokud je to v dané implementaci možné, nebo v kladnou konstantu typu `float`, která způsobí přetečení v době překladu.

V implementacích, které podporují „tichý NaN" pro typ `float`, by mělo být k dispozici makro `NAN`, které se rozvine v odpovídající hodnotu.

Ke klasifikaci čísel slouží makra `isnan()`, `isinf()`, `isnormal()`, `isfinite()` a `fpclassify()`. Smysl většiny z nich je jasný z názvu; tato makra vrací kladnou hodnotu, je-li jejich skutečným parametrem (v daném pořadí) NaN, nekonečno, normalizované reálné číslo, resp. konečné číslo. Makro `fpclassify()` vrací celočíselnou hodnotu určující, do které kategorie skutečný parametr patří. Možné hodnoty jsou definovány v `math.h` jako konstanty (opět vlastně makra) `FP_INFINITE`, `FP_NAN` a další.

PŘÍSTUP K VÝPOČETNÍMU PROSTŘEDÍ

V hlavičkovém souboru `fenv.h` najdeme nástroje, které umožňují pracovat s výpočetním prostředím pro reálná čísla, tj. zjišťovat a měnit ■

jeho nastavení. (To se týká jak nastavení požadovaných standardem IEC 60559, tak i dalších nastavení možných v dané architektuře.)

Datový typ `fenv_t` slouží k reprezentaci výpočetního prostředí pro reálná čísla jako celku. Hodnotu tohoto typu lze získat pomocí funkce `fegetenv()`, nastavené prostředí lze měnit pomocí `fesetenv()`. Typ `fexcept_t` reprezentuje příznaky, používané v tomto prostředí, jako celek. Pro jednotlivé příznaky jsou definována makra `FE_DIVIDEBYZERO`, `FE_INEXACT`, `FE_OVERFLOW` a další. Pro určení směru zaokrouhlení jsou k dispozici makra `FE_DOWNWARD`, `FE_TONEAREST` a další.

Stav výjimek lze zjistit pomocí funkce `fegetexceptflag()`, lze ho nastavit pomocí `fesetexceptflags()`. Další funkce umožňují výjimky vyvolávat, nastavovat jejich příznaky atd.

Funkce `fegetround()` zjistí nastavený způsob zaokrouhlování, funkce `fesetround()` ho nastaví.

Chceme-li v nějaké části programu pracovat s výpočetním prostředím pro reálná čísla, měli bychom o tom informovat překladač pomocí standardní direktivy

```
#pragma STDC FENV_ACCESS ON
```

a na konci tohoto úseku použít direktivu

```
#pragma STDC FENV_ACCESS OFF
```

Smyslem těchto direktiv je umožnit optimalizace, které mohou záviset na standardním nastavení výpočetního prostředí v dané architektuře.

Komplexní čísla

Nejprve si zopakujeme některé důležité pojmy. **Komplexní číslo** je, jak známo, vlastně dvojice reálných čísel. Máme-li komplexní číslo $z = (x, y)$, říkáme, že x je reálná část čísla z a y je jeho imaginární část. Číslo, jehož reálná část je nulová, označujeme jako imaginární.

Sčítání komplexních čísel je sčítání po složkách, tj. $(x, y) + (u, v) = (x+u, y+v)$. Podobně je definováno odečítání. Násobení je definováno složitěji, $(x, y) \times (u, v) = (xu-yv, xv+yu)$. Komplexní číslo **i** = (0, 1), pro něž platí rovnost $i^2 = -1$, označujeme jako **imaginární jednotku** (občas se místo **i** používá **j**.). Komplexní číslo $z = (x, y)$ často zapisujeme ve tvaru $z = x + iy$.

Typy pro komplexní čísla

Jazyk C nyní nabízí tři typy pro práci s komplexními čísly, a to `float _Complex`, `double _Complex` a `long double _Complex`. Jde pochopitelně o dvojice reálných čísel reprezentovaných pomocí typů `float`, `double`, resp. `long double`. O typu `float _Complex` říkáme, že jeho odpovídající reálný typ je `float`; podobně definujeme odpovídající reálný typ pro zbývající komplexní typy.

Vedle toho může implementace obsahovat tři datové typy pro reprezentaci imaginárních čísel, a to `float _Imaginary`, `double _Imaginary` a `long double _Imaginary`. Jsou to vlastně „obyčejná“ reálná čísla, chovají se ale jinak při aritmetických konverzích. I v případě imaginárních typů hovoříme o odpovídajících reálných typech.

Pro komplexní čísla jsou definovány základní aritmetické operace. Další nástroje pro počítání s komplexními čísly jsou definovány v hlavičkovém souboru `complex.h`.

Makra

V hlavičkovém souboru `complex.h` najdeme mj. makra `complex` a `imaginary` generující klíčová slova `_Complex` a `_Imaginary`. Makro `_Complex_I` se rozvine v hodnotu typu `const float _Complex` představující číslo **i**, makro `_Imaginary_I` se rozvine v hodnotu typu `const float _Imaginary` představující číslo **i**. (Makra `imaginary` a `_Imaginary_I` jsou k dispozici pouze v implementacích, které podporují typy `_Imaginary`.) Makro `I` se rozvine v `_Imaginary_I` nebo v `_Complex_I`.

Matematické funkce

V tomtéž hlavičkovém souboru najdeme také řadu matematických funkcí pro komplexní čísla. Jejich jména jsou podobná jako jména odpovídajících funkcí pro reálná čísla, ale začínají předponou „c“. Například funkce pro výpočet komplexního sinu má prototyp `double complex csin(double complex z)`. Vedle této funkce je k dispozici ještě funkce `csinf()` s parametrem typu `float complex`, vracející hodnotu téhož typu, a funkce `csinl()` s parametrem typu `long double complex`, jež vrací také `long double complex`.

Najdeme tu také funkce `creal()` a `cimag()` pro získání reálné, resp. imaginární části komplexního čísla, funkci `conj()` pro vytvoření komplexně sdruženého čísla a další.

Některé z těchto funkcí nejsou spojité v celé komplexní rovině – obsahují tzv. řez. Například funkce `csqrt()`, představující druhou odmocninu z komplexního čísla, má řez podél záporné reálné poloosy: Budeme-li se k nějakému bodu v tomto řezu blížit „shora“ (z oblasti s kladnou imaginární částí), budeme dostávat hodnoty výrazně jiné, než když se k témuž bodu budeme blížit „zdola“, z oblasti se zápornou imaginární částí. (Matematicky řečeno, v řezu je tato funkce nespojitá.) V implementacích, které obsahují kladnou a zápornou nulu, lze těchto hodnot použít k rozlišení „stran“ řezu.

Konverze

Reálná čísla lze konvertovat na komplexní a naopak. Při konverzi reálného čísla na komplexní číslo odpovídajícího typu (tedy např. při konverzi typu `double` na `double complex`) se z reálného čísla stane reálná část komplexního čísla a imaginární část výsledku bude 0. Podobně v implementacích, které obsahují imaginární typy, lze konvertovat hodnotu imaginárního typu na komplexní číslo odpovídajícího typu. Z imaginárního čísla se stane imaginární část výsledku, reálná část bude nulová.

Podobně lze konvertovat komplexní číslo na reálné nebo imaginární; při těchto konverzích se prostě „zahodí“ nepotřebná část komplexního čísla.

Při konverzích komplexních čísel se může pochopitelně měnit i „podkladový“ reálný typ. V takovém případě se konvertuje jak reálná, tak imaginární část podle pravidel pro konverze reálných čísel.

Obvyklé aritmetické konverze

Řada aritmetických operátorů očekává, že oba operandy budou stejného typu. Pokud jsou různých typů, proběhnou tzv. **obvyklé aritmetické konverze** *(usual aritmetic conversions)*, které oba operandy převedou na společný typ. Tento typ je pak zpravidla také typem výsledku.

To je stejné jako v předchozí verzi jazyka C. Ovšem pravidla pro tyto konverze se změnila; za prvé se berou v úvahu hodnosti celočíselných typů a za druhé se musí vypořádat s komplexními čísly. Následující **pravidla** se uplatňují v uvedeném pořadí:

- Odpovídá-li typ jednoho z operandů typu `long double`, převede se i druhý operand na typ odpovídající typu `long double`. (To znamená, že např. operand typu `double complex` se převede na `long double complex`.)
- Jinak odpovídá-li typ jednoho z operandů typu `double`, převede se i druhý operand na typ odpovídající typu `double`.
- Jinak odpovídá-li typ jednoho z operandů typu `float`, převede se i druhý operand na typ odpovídající typu `float`.

- Jinak se na oba operandy aplikují **celočíselná rozšíření** a pak tato **další pravidla**:
  - Jsou-li oba operandy již stejného typu, nejsou třeba žádné další konverze.
  - Jinak jsou-li oba operandy typu se znaménkem nebo jsou-li oba operandy typu bez znaménka, převede se operand s nižší hodností na typ s vyšší hodností.
  - Jinak má-li operand s typem bez znaménka stejnou nebo vyšší hodnost než typ druhého operandu, převede se operand znaménkového typu na typ druhého operandu (typ bez znaménka).
  - Jinak může-li typ se znaménkem reprezentovat všechny hodnoty typu bez znaménka, převede se operand „bezznaménkového" typu na typ se znaménkem.
  - Jinak se oba operandy převedou na typ bez znaménka odpovídající typu se znaménkem.

Tato pravidla vedou k podobným výsledkům jako pravidla v předchozí verzi standardu, umožňují ovšem počítat i s dalšími celočíselnými datovými typy.

Příště

Příště povídání o novinkách v posledním standardu jazyka C dokončíme. Zbývá nám ještě podívat se zejména na ukazatele, pole, funkce a makra. Ve stručnosti si všimneme ještě také změn ve standardní knihovně. ■ ■ ■ *Miroslav Virius, autor@chip.cz*

Reálná čísla v počítači

Podívejme se krátce, jak se v počítači obvykle zobrazují reálná čísla. Vyjdeme od zápisu, který používáme pro velmi velká nebo velmi malá čísla, např. $x = 2{,}654 \times 10^{14}$. Část 2,654 označíme jako **mantisu**, číslo 14 jako (desítkový) **exponent**.

Takovéto vyjádření ovšem není jediné možné – totéž číslo lze vyjádřit řadou různých zápisů. Například uvedené číslo x lze zapsat také jako $26{,}54 \times 10^{13}$ nebo $0{,}2654 \times 10^{15}$ atd. Proto se používá ještě dodatečná podmínka, že mantisa m nenulového čísla musí ležet např. v rozmezí $1 \le m < z$, kde z je základ číselné soustavy. (Stejně dobře by bylo možné použít podmínku $z^{-1} \le m < 1$.) Číslo v tomto tvaru označujeme jako **normalizované**.

Podobný zápis, ovšem založený na dvojkové soustavě, je základem zobrazení reálných čísel v počítači. Vezmeme-li nenulové číslo y a zapíšeme-li ho ve dvojkové soustavě v normalizovaném tvaru, např. $y = 1{,}00111 \times 2^{10011}$, vidíme, že stačí ukládat mantisu jako celé číslo (tj. bez řádové čárky, neboť ta je vždy na stejném místě) a exponent. Ve skutečnosti je skupina bajtů sloužících k uložení reálného čísla zpravidla rozdělena na tři pole. V prvním, jednobitovém, se ukládá znaménko mantisy, v dalších absolutní hodnota mantisy a exponent, zpravidla zvětšený o nějakou konstantu (tzv. posunutí) umožňující vyjádřit záporné znaménko ještě v oblasti kladných hodnot – viz dále. Je jasné, že šířka (počet bitů) mantisy určuje přesnost (počet platných cifer), šířka exponentu určuje řád největšího a nejmenšího čísla, které lze zobrazit.

Jeden bit mantisy lze někdy ušetřit a tak zvětšit přesnost: nenulové číslo v normalizovaném tvaru obsahuje vždy před řádovou čárkou číslici 1, a proto ji není třeba ukládat. Ne vždy se to však používá; např. na PC se této úspory využívá pro čtyřbajtová a osmibajtová reálná čísla (jež odpovídají typům `float` a `double`), nikoli však pro 10bajtová reálná čísla, která v některých implementacích odpovídají typu `long double`.

Obsahuje-li znaménková část, mantisa a exponent po řadě hodnoty s, m a e, může pak být hodnota uloženého nenulového čísla vyjádřena zápisem $(-1)^s \times 1{,}m \times 2^{e-q}$, kde q je **posunutí** *(bias)*. Přitom $1{,}m$ představuje číslo, které vznikne zápisem jedničky, za níž následuje řádová čárka a poté bity mantisy.

Posunutí q se volí tak, aby byly všechny hodnoty e kladné. To umožňuje vyhradit některé hodnoty exponentu pro zvláštní účely – např. pro zobrazení 0, nekonečna nebo hodnoty označované NaN, o nichž si povíme dále.

Je jasné, že v počítači lze zobrazit pouze konečnou podmnožinu množiny reálných čísel. Čísla, která nelze zobrazit, jsou buď příliš velká, nebo leží mezi hodnotami, které zobrazit lze. V prvním případě dojde k přetečení, ve druhém případě se číslo nějakým způsobem zaokrouhlí na jednu ze sousedních zobrazitelných hodnot. (Nemusí to být nejbližší; často lze volit, zda se má zaokrouhlovat směrem k nejbližšímu číslu, směrem k nule, směrem k plus nebo minus nekonečnu atd.)

Nekonečno

Standard IEC 60559 určuje, že jednou z možných hodnot, které lze zobrazit, má být i **kladné** a **záporné nekonečno;** vznikne např. jako výsledek dělení nenulového čísla nulou. Platí, že $+\infty$ je větší než jakékoli jiné reálné číslo; přičtením konečného čísla k nekonečnu dostaneme opět nekonečno atd.

S nekonečnem lze tedy do jisté míry počítat podobně jako s „obyčejnými" čísly; operace $\infty - \infty$, $+\infty \times 0$ a ∞/∞ ovšem nemají smysl.

NaN

Další ze zvláštních hodnot, které by měly být reprezentovatelné v reálných číslech, je tzv. **NaN**. Toto označení vzniklo jako zkratka pro *Not a Number*, tedy něco, co není číslo. NaN vznikne jako výsledek operací, které nemají smysl – např. při násobení $+\infty \times 0$.

Standard IEC 60559 rozlišuje **tichý** *(quiet)* a **signalizující** *(signaling)* NaN. Tichý NaN představuje možný výsledek, signalizující NaN způsobí výjimku (tedy typicky přerušení výpočtu a ohlášení chyby).

Denormální čísla

Může se stát, že výsledek aritmetické operace nebude nulový, bude ale mít tak malou absolutní hodnotu, že ho nepůjde v daném datovém typu vyjádřit v normalizovaném tvaru. Některé architektury počítačů umožňují pracovat i s takovýmito **denormálními** čísly (tj. nezaokrouhlí je na 0 nebo na nejmenší nenulové zobrazitelné číslo). Denormální čísla jsou ovšem méně přesná, tj. mají menší počet platných cifer než normalizovaná čísla.

Výjimky

Některé operace nebo jejich výsledky mohou způsobit výjimky procesoru. Jde např. o dělení nulou, vznik denormálního čísla, vznik signalizujícího NaN atd. To může, ale nemusí znamenat přerušení výpočtu – záleží na nastavení výpočetního prostředí *(floating point execution environment)*.

Výpočetní prostředí

Pod tímto abstraktním označením se skrývají mechanismy, které umožňují ovlivňovat chování procesoru při výpočtech – např. směr zaokrouhlování, způsob reakce na některé výjimky atd. Znáte-li architekturu PC, můžete si pod tímto označením představit analogie řídicího a stavového slova matematického koprocesoru.