KAPITOLA 3

Datové typy pro reálná čísla

V této kapitole:

- Vlastnosti datových typů pro reálná čísla v jazyce C/C++
- Vstupně/výstupní operace z pohledu reálných čísel
- Aritmetické operace s reálnými čísly
- Implicitní a explicitní typové konverze
- Priorita a asociativita dosud probraných operátorů
- Funkce z matematické knihovny

V kapitole 2 jsme se seznámili s celočíselnými datovými typy. Pro uložení číselných hodnot ve velkém rozsahu obvykle nepožadujeme tak velkou přesnost, jakou nám poskytují celá čísla. Obvykle vystačíme s přesností například 6 desítkových číslic.

Pro reprezentaci reálných čísel pak používáme datové typy obecně označované jako *čísla v plovoucí řádové čárce* (anglicky *float-point*). Takové číslo je reprezentováno dvěma složkami: mantisou a exponentem. *Mantisa* (M) je hodnota čísla, *exponent* (E) určuje řád. Viz tento předpis (pro desítkovou soustavu):

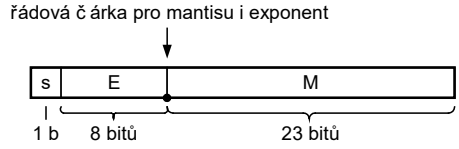
$$\text{reálné číslo} = M \cdot 10^{E}$$

Příklady zápisů čísel v tomto formátu: $1{,}602 \cdot 10^{-19}$ (elementární náboj); $6{,}023 \cdot 10^{23}$ (Avogadrova konstanta).

Standard IEEE 754

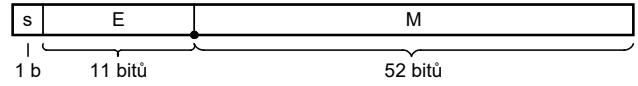
Tento standard byl vyvinut již roku 1985 a stal se obecným standardem pro čísla v plovoucí řádové čárce. Postupně bylo zavedeno několik formátů čísel v plovoucí řádové čárce. Běžně se používají formáty šíře 16, 32, 64, 80, 128, 256 bitů. Na úrovni jazyka C/C++ se pak používají formáty označované jako **single**, **double** a **extended**:

- **Single** – jednoduchá přesnost, šíře 32 bitů, hodnotu čísla stanovíme pomocí vzorce: $A = (-1)^S \cdot 1,M \cdot 2^{E-127}$.



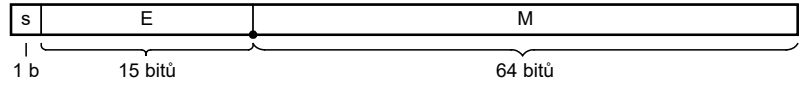
Obrázek 3.1 Formát 32bitového čísla v plovoucí řádové čárce (single)

- **Double** – dvojnásobná přesnost, šíře 64 bitů, hodnotu čísla stanovíme pomocí vzorce: $A = (-1)^S \cdot 1,M \cdot 2^{E-1023}$.



Obrázek 3.2 Formát 64bitového čísla v plovoucí řádové čárce (double)

- **Extended** – rozšířená dvojnásobná přesnost, šíře 80 bitů, hodnotu čísla stanovíme pomocí vzorce: $A = (-1)^S \cdot 1,M \cdot 2^{E-16\,384}$.



Obrázek 3.3 Formát 80bitového čísla v plovoucí řádové čárce (extended)

V obrázcích 3.1 až 3.3 mají symboly následující význam:

- **s** – znaménko čísla,
- **E** – exponent uložený v posunutém kódu,
- **M** – mantisa uložená v kódu se skrytou jedničkou (celá mantisa je 1,M), tímto způsobem se rozsah mantisy zvýší o jeden bit.

Některé kombinace jsou vyhrazeny pro reprezentaci zvláštních stavů, například:

- nekonečno – E = samé 1, M = samé 0, s = 0/1 (kladné i záporné nekonečno),
- NaN (Not A Number, česky *není číslo*) – E = samé 1, M = nenulové číslo, používá se pro uložení výsledku pro případy operací: 0/0, ∞/∞, √(–*1*), ...

Realizace operací s čísly v plovoucí řádové čárce vyžaduje značnou výpočetní podporu.

Připomeňme, že jednodušší procesory nemají podporu pro operace s čísly v plovoucí řádové čárce implementovánu na hardwarové úrovni. Veškeré výpočty s těmito čísly jsou pak realizovány programově a představují pro procesor nezanedbatelné zpoždění.

Naproti tomu procesory doplněné koprocesorem nebo procesory s integrovanou jednotkou FPU (float-point unit, jednotka pro výpočty v plovoucí řádové čárce) mají pro práci s čísly

v plovoucí řádové čárce hardwarovou podporu, takže doba výpočtu může být srovnatelná s celočíselnou aritmetikou.

Vlastnosti datových typů pro reálná čísla v jazyce C/C++

K dispozici jsou tři datové typy pro operace s čísly v plovoucí řádové čárce: **`float`**, **`double`** a **`long double`**, které se liší oborem hodnot. Tyto typy se také liší velikostí v bajtech a přesností, která je určena počtem platných cifer (číslic).

Tabulka 3.1 Vlastnosti datových typů pro reálná čísla

Typ	Přibližný obor hodnot
float odpovídá **single** (viz obr. 3.1), velikost: 4 bajty, přesnost: 6 platných desítkových cifer	$\pm 1{,}2\times10^{-38}$ až $\pm 3{,}4\times10^{+38}$
double odpovídá **double** (viz obr. 3.2), velikost: 8 bajtů, přesnost: 15 platných desítkových cifer	$\pm 2{,}2\times10^{-308}$ až $\pm 1{,}8\times10^{+308}$
long double odpovídá **extended** (viz obr. 3.3), velikost: 16 bajtů, přesnost: 18 platných desítkových cifer	$\pm 3{,}4\times10^{-4932}$ až $\pm 1{,}2\times10^{+4932}$

Charakteristiky těchto typů jsou stručně shrnuty v tabulce 3.1. Jelikož jsou mantisa a exponent uloženy v binární podobě, je uváděn pouze přibližný obor hodnot v desítkové soustavě. Podobně uváděná přesnost mantisy v počtu desítkových cifer je zaokrouhlena dolů na nejbližší celé číslo. Rovněž můžeme vysledovat jistou nesymetrii oboru hodnot pro nejnižší a nejvyšší hodnoty.

Poněkud zvláštní je typ **`long double`**. Řada implementací jazyka C/C++ používá pro realizaci tohoto typu rozšířenou dvojitou přesnost (extended), která historicky pochází z koprocesoru Intel 8087. Tento typ standardně zabírá 80 bitů, tedy 10 bajtů. Z principiálních důvodů je však pro proměnnou **`long double`** tradičně alokováno ne 10, ale 16 bajtů. Důvodem je tzv. zarovnávání paměti. Přístup procesoru do datové paměti totiž probíhá obvykle v blocích, které jsou násobkem 4 bajtů.

Některé překladače řeší tento problém implementací dvou variant pro reálná čísla v přesnosti větší než **`double`**. Například se můžeme setkat s typy **`__float80`** a **`__float128`**. Typ **`__float80`** odpovídá přesnosti **extended**, přesto však v paměti zabírá 16 bajtů. Typ **`__float128`** odpovídá přesnosti **quadruple** (čtyřnásobná přesnost), v paměti zabírá také 16 bajtů.

Zápis literálů a charakteristiky typů

Formát zápisu reálných literálů v desetinném tvaru se vyznačuje používáním tečky místo obvyklé desetinné čárky, například: **`1.2345`**. Reálné literály lze také zapisovat v semilogaritmickém tvaru, například: **`1.602E-19`**.

Literály reálných čísel jsou standardně brány ve dvojnásobné přesnosti (double). Pro rozlišení literálů v různé přesnosti používáme přípony: **`F`** nebo **`f`** pro typ **`float`**, **`D`** nebo **`d`** pro typ **`double`** a **`L`** nebo **`l`** pro typ **`long double`**.

Přesné charakteristiky datových typů pro čísla v plovoucí řádové čárce lze získat pomocí symbolů z hlavičkového souboru **FLOAT.H** (resp. CFLOAT v jazyce C++). Vybrané symboly jsou uvedeny v tabulce 3.2.

Tabulka 3.2 Vybrané charakteristiky čísel v plovoucí řádové čárce

Symbol	Význam
`FLT_DIG`	počet platných desítkových cifer typu **`float`**
`FLT_MIN`	minimální hodnota typu **`float`**
`FLT_MAX`	maximální hodnota typu **`float`**
`DBL_DIG`	počet platných desítkových cifer typu **`double`**
`DBL_MIN`	minimální hodnota typu **`double`**
`DBL_MAX`	maximální hodnota typu **`double`**
`LDBL_DIG`	počet platných desítkových cifer typu **`long double`**
`LDBL_MIN`	minimální hodnota typu **`long double`**
`LDBL_MAX`	maximální hodnota typu **`long double`**

Níže je doplněn krátký program, který pro každý z uvedených typů vypíše postupně: velikost v bajtech, počet platných desítkových číslic, minimální a maximální hodnotu. Můžete tyto údaje porovnat s tabulkou 3.1.

Dále je deklarována globální konstanta **`pi`** (není deklarována v rámci žádné funkce, proto je k dispozici v celém zdrojovém souboru). Hodnota této konstanty odpovídá Ludolfovu číslu π.

Nakonec jsou vypsány velikosti typů **`__float80`** a **`__float128`**.

PROG_01:

```
#include <iostream>
#include <float.h>
using namespace std;

const double pi=3.14159265;
int main(int argc, char** argv)
{
  cout<<"sizeof(float)="<<sizeof(float)<<endl;
  cout<<"FLT_DIG="<<FLT_DIG<<endl;
  cout<<"FLT_MIN="<<FLT_MIN<<endl;
```

```
  cout<<"FLT_MAX="<<FLT_MAX<<endl;
  cout<<endl;

  cout<<"sizeof(double)="<<sizeof(double)<<endl;
  cout<<"DBL_DIG="<<DBL_DIG<<endl;
  cout<<"DBL_MIN="<<DBL_MIN<<endl;
  cout<<"DBL_MAX="<<DBL_MAX<<endl;
  cout<<endl;

  cout<<"sizeof(long double)="<<sizeof(long double)<<endl;
  cout<<"LDBL_DIG="<<LDBL_DIG<<endl;
  cout<<"LDBL_MIN="<<LDBL_MIN<<endl;
  cout<<"LDBL_MAX="<<LDBL_MAX<<endl;
  cout<<endl;

  cout<<"pi="<<pi<<endl;

  cout<<"sizeof(__float80)="<<sizeof(__float80)<<endl;
  cout<<"sizeof(__float128)="<<sizeof(__float128)<<endl;

  return 0;
}
```

Výpis programu v konzoli:

```
sizeof(float)=4
FLT_DIG=6
FLT_MIN=1.17549e-038
FLT_MAX=3.40282e+038

sizeof(double)=8
DBL_DIG=15
DBL_MIN=2.22507e-308
DBL_MAX=1.79769e+308

sizeof(long double)=16
LDBL_DIG=18
LDBL_MIN=3.3621e-4932
LDBL_MAX=1.18973e+4932

pi=3.14159
sizeof(__float80)=16
sizeof(__float128)=16
```

Poznámka: Výpis proměnné `pi` je zaokrouhlen na 6 cifer přesto, že typ `double` má přesnost 15 cifer. Jedná se pouze o výchozí nastavení formátu pro výpis reálných čísel. Níže se dozvíme, jak lze formát výpisu upravit.

Vstupně-výstupní operace z pohledu reálných čísel

Informace ke vstupně-výstupním operacím z kapitoly 2 můžeme nyní rozšířit o přehled dalších manipulátorů a doplnit informacemi, které platí pro reálná čísla.

Při výpisu hodnoty můžeme nastavit parametry, které určí, jak přesně výpis proběhne:

- **šířka** – určuje minimální počet znaků, které se mají vypsat. Uvažujme například výpis čísla 56 na 5 míst. V tomto případě je třeba přidat další 3 znaky, aby celková šířka výpisu byla 5 znaků. Znak, který se použije jako výplň, je obvykle mezera (je možné jej ale změnit pomocí parametru *výplňový znak*). Pokud uvažujeme výpis čísla 56 v šířce 0, vypíše se prostě `56` (výpis hodnoty nesmí být zkreslen). Šířku nastavuje manipulátor **`setw`**.
- **přesnost** – určuje počet číslic za desetinnou tečkou, které se zobrazí. Například výpis čísla π (3,14159265 …) při šířce 5 a přesnosti 3 vypadá takto: `3.142` (při výpisu probíhá nezbytné zaokrouhlení); platí pro formáty **`fixed`** a **`scientific`**. Přesnost nastavuje manipulátor **`setprecision`**.
- **výplňový znak** – určuje znak, který se použije jako výplň při výpisu hodnoty ve větší šířce; jako výchozí výplňový znak je použita mezera. Výplňový znak nastavuje manipulátor **`setfill`**.
- **zarovnávání** – určuje způsob zarovnání výpisu hodnoty a místo, kam se budou vkládat výplňové znaky. Rozlišujeme (uvažujme výpis čísla -1 při šířce 5):
  - zarovnání doleva (**`left`**) – výpis hodnoty je zarovnán doleva, výplň se vkládá zprava: `-1□□□`,
  - zarovnání doprava (**`right`**) – výpis hodnoty je zarovnán doprava, výplň se vkládá zleva: `□□□-1`,
  - mezi znaménko a hodnotu (**`internal`**) – výpis znaménka je vlevo, následují výplňové znaky a nakonec vypisovaná hodnota: `-□□□1`.

Přehled běžně používaných manipulátorů pro výpis uvádí **tabulka 3.3**. Na začátku jsou připomenuty dříve popsané manipulátory **`endl`**, **`dec`**, **`oct`** a **`hex`**.

Tabulka 3.3 Přehled manipulátorů

Manipulátor	Použitelnost pro čísla	Význam
`endl`	celá/reálná	ukončí řádek a nastaví kurzor na začátek následujícího řádku
`dec`	celá	přepne zobrazení celých čísel do desítkové soustavy
`oct`	celá	přepne zobrazení celých čísel do osmičkové soustavy
`hex`	celá	přepne zobrazení celých čísel do šestnáctkové soustavy
`showpos`	celá/reálná	zajistí zobrazení znaménka i pro nezáporná čísla (kladná čísla včetně nuly)

Manipulátor	Použitelnost pro čísla	Význam
noshowpos	celá/reálná	vypne zobrazení znaménka pro kladná čísla (znaménko se zobrazuje pouze pro záporná čísla)
left	celá/reálná	přepne na zarovnávání doleva (výplň zprava)
right	celá/reálná	přepne na zarovnávání doprava (výplň zleva)
internal	celá/reálná	přepne na vyplnění mezi znaménkem a hodnotou
fixed	reálná	přepne výpis reálných čísel do tvaru s pevnou pozicí desetinné tečky
scientific	reálná	přepne výpis reálných čísel do exponenciálního tvaru
uppercase	celá/reálná	přepne výpis čísel tak, že znaky se zobrazují jako velká písmena (týká se výpisu celých čísel v šestnáctkové soustavě a reálných čísel v exponenciálním tvaru)
nouppercase	celá/reálná	přepne výpis čísel tak, že znaky se zobrazují jako malá písmena (týká se výpisu celých čísel v šestnáctkové soustavě a reálných čísel v exponenciálním tvaru)
setw(w)	celá/reálná	nastaví šířku následujícího výpisu dle **w**, chybějící znaky do požadovaného počtu jsou realizovány zvolenou výplní (pokud zadání šířky nezopakujeme před každým výpisem, bude mít následující výpis opět šířku 0)
setfill(f)	celá/reálná	nastaví výplňový znak na **f** (znakové literáry píšeme mezi apostrofy)
setprecision(p)	reálná	nastaví výpis reálných čísel na počet desetinných míst daných **p**

Poslední tři manipulátory jsou parametrické (jsou řízeny parametrem zapsaným v závorkách). Pro použití těchto manipulátorů je nutné do zdrojového textu vložit hlavičkový soubor **iomanip**:

```
#include <iomanip>
```

Pro lepší pochopení připojujeme několik příkladů použití výše popsaných manipulátorů pomocí tabulky 3.4.

Tabulka 3.4 Příklady formátování výpisu

Příklad	Výpis	Vysvětlení
cout<<56;	56	normální výpis kladné hodnoty (bez znaménka +)
cout<<-56;	-56	výpis záporné hodnoty musí vždy obsahovat znaménko –
cout<<showpos<<56;	+56	výpis kladné hodnoty včetně znaménka
cout<<setw(5)<<left<<-56;	-56□□	šířka 5, zarovnání doleva: po výpisu čísla se připojí 2 mezery
cout<<setw(5)<<right<<-56;	□□-56	šířka 5, zarovnání doprava: před výpisem čísla se vloží 2 mezery
cout<<setw(5)<<internal<<-56;	-□□56	šířka 5, zarovnání mezi: mezi znaménko a číslice se vloží 2 mezery

Příklad	Výpis	Vysvětlení
`cout<<fixed<<3.141519265;`	`3.141519`	výpis reálného čísla v desetinném tvaru
`cout<<fixed<<setprecision(3) <<3.141519265;`	`3.142`	zobrazí se 3 číslice za desetinnou tečkou (+zaokrouhlení)
`cout<<scientific<<setprecision(3) <<3.141519265;`	`3.142e+000`	výpis v exponenciálním tvaru, zobrazí se 3 číslice za desetinnou tečkou (+zaokrouhlení)
`cout<<setw(10)<<left<<setfill('0') <<fixed<<setprecision(3) <<3.141519265;`	`3.14200000`	výpis v desetinném tvaru, 3 platné číslice za desetinnou tečkou, zbytek se doplní nulami do celkové šíře 10 znaků (zarovnání doleva)
`cout<<setw(10)<<right<<setfill('0') <<fixed<<setprecision(3) <<3.141519265;`	`000003.142`	výpis v desetinném tvaru, 3 platné číslice za desetinnou tečkou, vlastní výpis předchází nuly do celkové šíře 10 znaků (zarovnání doprava)

Aritmetické operace s reálnými čísly

Pro reálná čísla lze používat běžné aritmetické operátory stejné jako pro celá čísla. Jsou zde však dva drobné rozdíly:

1. Operátor **%** (modulo, zbytek po celočíselném dělení) nemá pro aritmetiku reálných čísel smysl, jeho použití s reálným číslem je chápáno jako chyba.
2. Operátor **/** (dělení) má v reálné aritmetice smysl reálného dělení, tedy například výraz `5.0/2.0` má výsledek 2,5.

Přehled operátorů použitelných pro reálná čísla je uveden v tabulkách 3.5 a 3.6. V tabulce 3.5 jsou běžné binární operátory (mají levý a pravý operand), v tabulce 3.6 jsou unární operátory (mají pouze jeden operand).

Tabulka 3.5 Základní aritmetické operátory

Operátor	Příklad zápisu	Význam
+	**x+y**	součet dvou čísel uložených v proměnných x a y
–	**x–y**	rozdíl dvou čísel uložených v proměnných x a y
*****	**x*y**	součin dvou čísel uložených v proměnných x a y
/	**x/y**	podíl dvou čísel uložených v proměnných x a y

Tabulka 3.6 Unární aritmetické operátory

Operátor	Příklad zápisu	Význam
+	**+x**	kladné znaménko čísla
–	**–y**	záporné znaménko čísla (násobení –1)
++	**x++ nebo ++x**	**x++** postinkrement, **++x** preinkrement
––	**x–– nebo ––x**	**x––** postdekrement, **––x** predekrement

Implicitní a explicitní typové konverze

Při práci s různými datovými typy se často nevyhneme konstrukci, kde v rámci jednoho výrazu použijeme proměnné nebo literáry různých číselných typů (doplníme nyní informace ke kapitole 2). Vznikají pak tyto otázky:

- Jak překladač tento problém řeší?
- V jaké aritmetice se počítá?
- Může dojít k chybnému vyhodnocení výrazu?

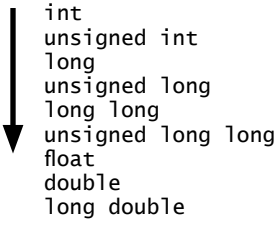
Překladač může provádět dva základní typy převodů (konverzí) dat:

- **Implicitní typová konverze** – slovem implicitní označujeme případ, kdy k převodu dochází automaticky; překladač použije implicitní převod v případě, kdy je to nezbytně nutné.
- **Explicitní typová konverze** – slovem explicitní označujeme případ, kdy je převod vynucen programátorem tak, že ho v programu předepíše pomocí operátoru přetypování.

Implicitní typové konverze

K implicitní typové konverzi dochází v níže uvedených případech:

1. Konverze samostatných operandů: Před vyhodnocením výrazu se samostatné operandy převádějí takto:
   - **`char`** nebo **`short`** se konvertují na **`int`**,
   - **`unsigned char`** nebo **`unsigned short`** se konvertují na **`int`** (pokud **`int`** může reprezentovat jejich hodnotu, tzn. nepřeteče) nebo na **`unsigned int`** (pokud se nezdařila konverze na **`int`**).
2. Binární operace: Pokud mají operandy binární operace různý typ, konvertuje se typ operandu s nižší prioritou na typ operandu s vyšší prioritou. Priorita je pevně dána (typy **`char`** a **`short`** se nejdříve konvertují na **`int`**), viz obrázek 3.4 (typ **`int`** má nejnižší prioritu):



Obrázek 3.4 Priorita pro konverzi operandů binární operace

3. Převod vynucený přiřazením: V přiřazovacích příkazech je typ na pravé straně konvertován na typ na levé straně. Pokud překladač takovou konverzi nedokáže provést, označí daný řádek programu jako chybný.

Vysvětlující příklad – Uvažujeme tyto proměnné:

```
int a;
float b=3.3;
short c=2;
```

A nyní, jak se vyhodnotí výraz:

```
a=b*c;
```

Bude se postupovat v těchto krocích:

1. Použije se pravidlo pro samostatné operandy. Operand **b** je typu **short** a proto se převede na hodnotu typu **int**. Operand **c** je typu **float** a převádět se nebude.
2. Operátorem * je předepsána binární operace. Operand **c** je typu **float** a má tak vyšší prioritu. Proto se operand **b** dříve převedený na typ **int** převede ještě jednou a to na typ **float**. Výsledkem operace je hodnota **6.6**.
3. Nyní se výsledek výrazu přiřazuje do proměnné **a**, která je typu **int**. Použije se pravidlo pro přiřazení. Reálná hodnota **6.6** se převede na celé číslo tak, že se odsekne desetinná část (pozor: nezaokrouhluje se!). Výsledek je tedy **6** a je to hodnota typu **int**.

Je-li třeba provést při převodu z reálného čísla na celé zaokrouhlení, přičteme k zaokrouhlovanému číslu 0,5. Například: **a=b*c+0.5**;. Hodnota 6,6 je tedy zvýšena na 7,1 a po odseknuté desetinné části je výsledek zaokrouhlen na 7.

Možné problémy implicitních převodů

Převod z nižšího na vyšší typ (tzv. povýšení) je bez problémů. Nedochází ke ztrátě uložené informace, pouze se změní datový typ uložené hodnoty.

Problémy přináší převod z vyššího typu na nižší typ. Zde může v některých případech dojít ke ztrátě (zkreslení) údaje. Uvažujme níže uvedený příklad.

PROG_02:

```
#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
  int a=123456;
  short b=a;
  cout<<b<<endl;

  return 0;
}
```

Výpis programu v konzoli:

```
-7616
```