

Komplexní čísla

1205 Základní pojmy



Než se pustíme do tipů ukazujících práci s komplexními čísly, připomeneme si, o co jde. Komplexní číslo je vlastně uspořádaní dvojice reálných čísel. Její první složku nazýváme *reálná část* a druhou složku *imaginární část*. Zapisujeme je několika způsoby. První z nich je *souřadnicová* reprezentace, která spočívá v tom, že reálnou a imaginární část uzavřeme do závorek a oddělíme je čárkou: $z = (a, b)$.

Číslo, jehož reálná část je nulová, označujeme jako (ryze) *imaginární*. Číslo $(0, 1)$ označujeme symbolem i a nazýváme ho *imaginární jednotka*. Použijeme-li toto označení, můžeme jakékoli komplexní číslo $z = (a, b)$ zapsat v *algebraickém* tvaru $z = a + i.b$.

Pro komplexní čísla je definováno sčítání a odečítání po složkách – reálná část součtu je součtem reálných částí, imaginární část součtu je součtem imaginárních částí. Ze vztahu

$$i^2 = -1$$

lze odvodit vzorce pro násobení a dělení komplexních čísel.

1206 Datové typy pro komplexní čísla v C99 a odpovídající reálné typy



Jazyk C99 nabízí 3 datové typy pro komplexní čísla, a to `float _Complex`, `double _Complex` a `long double _Complex`. Jde pochopitelně o dvojice reálných čísel typů `float`, `double` a `long double`.

O typu `float _Complex` říkáme, že jeho *odpovídající reálný typ* je `float`; podobně definujeme *odpovídající reálný typ* pro zbývající komplexní typy.

1207 Imaginární čísla (C99)



Implementace může také nabízet datové typy pro reprezentaci imaginárních čísel, a to `float _Imaginary`, `double _Imaginary` a `long double _Imaginary`. Imaginární číslo je reprezentováno jedním reálným číslem, chová se ale jinak při aritmetických konverzích. I v případě imaginárních typů hovoříme o *odpovídajících reálných typech*.

1208 Základní matematické operace s komplexními čísly



Pro komplexní čísla jsou definovány binární aritmetické operátory $+$, $-$, $*$ a $/$. Můžeme pro ně také použít unární operátory $+$ a $-$.

K porovnávání komplexních čísel lze použít relační operátory `==` a `!=`; dvě komplexní čísla si jsou rovna, rovnají-li se jejich reálné i imaginární části. Vzhledem k implicitním konverzím lze porovnávat také reálná čísla s komplexními.

Operace „menší než“, „menší nebo rovno“ atd. nejsou pro komplexní čísla definovány, takže odpovídající relační operátory pro ně použít nesmíme. Nejsou pro ně také definovány operátory `++` a `--`.

1209 Imaginární jednotka (C99)



V hlavičkovém souboru `<complex.h>` najdeme makro `_Complex_I`, které se rozvine v imaginární jednotku. V implementacích, které podporují typy pro imaginární čísla, najdeme také makro `_Imaginary_I`, jež se rozvine v imaginární jednotku typu `float _Imaginary`.

Makro `I` se rozvine buď v `_Imaginary_I`, nebo v `_Complex_I`.

Všechna tato makra můžeme v programu zrušit direktivou `#undef`.

1210 Alternativní modifikátory (C99)



Podivně vypadající modifikátory `_Complex` a `_Imaginary`, jež odporují tradicím jazyka C, byly zavedeny především proto, aby pokud možno nedošlo ke konfliktům s různými proprietárními rozšířeními starších překladačů. Proto je v hlavičkovém souboru `<complex.h>` definováno makro `complex`, které se rozvine v klíčové slovo `_Complex`. V implementacích, které definují klíčové slovo `_Imaginary`, je také definováno makro `imaginary`, které se rozvine v `_Imaginary`.

Program ovšem může tato makra zrušit direktivou `#undef`.

1211 Deklarace proměnné s inicializací (C99)



Chceme-li deklarovat proměnnou některého z typů pro komplexní čísla a zároveň ji inicializovat, můžeme využít makro `I`:

```
double complex z = a + b*I;
```

Můžeme samozřejmě také použít jinou proměnnou

```
double complex d = z;
```

Tato proměnná může mít i jiný *odpovídající reálný typ* než deklarovaná proměnná.

1212 Reálná a imaginární část (C99)



Máme komplexní číslo `z` a chceme ho rozložit na reálnou a imaginární část. K tomu nám poslouží funkce `creal()` a `cimag()`.

```
double x = creal(z);
```

Tato funkce očekává parametr typu `double complex` a vrací hodnotu typu `double`. Vedle toho máme k dispozici ještě funkce `crealf()`, resp. `creall()` a `cimagf()`, resp. `cimagl()` pro typy `float complex`, resp. `long double complex`.

1213 Vstup a výstup komplexních čísel (C99)



Jazyk C99, jak se zdá, vstup a výstup komplexních čísel neřeší. To znamená, že musíme vypisovat, resp. načítat zvlášť reálnou a imaginární část komplexního čísla. Například takto:

```
printf("%f + i.%f", creal(z), cimag(z));
```

1214 Komplexně sdružené číslo (C99)



Komplexně sdružené číslo k číslu $z = a + i.b$ je číslo $w = a - i.b$. V C99 ho získáme pomocí funkce `conj()`, jež očekává parametr typu `double complex` a vrací hodnotu téhož typu. Vedle toho máme k dispozici funkce `conjf()`, resp. `conjl()` pro zbývající dva typy komplexních čísel.

1215 Konverze mezi typy komplexních čísel (C99)



Hodnotu typu představujícího komplexní číslo lze automaticky konvertovat na hodnotu typu představujícího komplexní číslo s jiným *odpovídajícím reálným typem* (viz tip 1206). To znamená, že platí-li deklarace

```
double complex z;  
float complex w;
```

můžeme bez problémů napsat např.

```
z = w;
```

1216 Konverze mezi komplexními, reálnými a ryze imaginárními čísly (C99)



Komplexní číslo může být automaticky konvertováno na hodnotu *odpovídajícího reálného typu*. Přitom se „zahodí“ imaginární část. Automatická je i opačná konverze, při níž vznikne z hodnoty reálného typu komplexní číslo s nulovou imaginární částí.

V implementacích, které podporují imaginární čísla, je také možná konverze komplexního čísla na imaginární číslo; při ní se „zahodí“ reálná část. Platí-li deklarace

```
float complex z = 3 + 4*I;
```

```
float imaginary x;
```

způsobí přiřazení

$x = z$;

že se do x uloží hodnota 4. Při opačné konverzi, která je také automatická, vznikne komplexní číslo s nulovou reálnou částí.

1217 Matematické funkce (C99)



Pro komplexní čísla je definována řada matematických funkcí; jedná se o tytéž funkce, které jsou definovány pro reálné typy. Jejich jména dostaneme tak, že ke jménu funkce pro reálná čísla připojíme úvodní „c“. To znamená, že např. analogií funkce `atan()` je funkce `catan()` atd. Všechny tyto funkce jsou definovány ve třech variantách, a to pro typ `double complex`, `float complex` a `long double complex`. Varianty pro poslední dva typy končí „f“, resp. „l“, takže např. varianta funkce `arkustangens` pro typ `long double complex` se jmenuje `catanl()`.



Upozornění: Následující tři tipy, nadepsané „Poznámky k ...“, nevyčerpávají všechny matematické funkce definované pro komplexní čísla. Pouze upozorňují na zvláštnosti definičního oboru funkcí, u nichž je to třeba. Nehovoříme zde o funkcích, jako je `sin()`, `exp()` a další, protože o nich prostě není třeba hovořit.

1218 Poznámky k inverzním trigonometrickým funkcím pro komplexní čísla (C99)



I když zde hovoříme pouze o funkcích pro typ `double complex`, platí uvedené poznámky pro funkce pro všechny komplexní typy.

- Funkce `cacos()` a `casin()`, jež počítají komplexní arkuskosinus, resp. arkussinus, mají řez podél reálné osy mimo interval $\langle -1, 1 \rangle$. Pro reálnou část argumentu platí stejné omezení jako v případě funkcí reálné proměnné, imaginární část není omezena.
- Funkce `catan()`, jež počítá komplexní arkustangens, má řez podél imaginární osy s výjimkou intervalu $\langle -i, i \rangle$. Pro reálnou část argumentu platí stejné omezení jako v případě funkce reálné proměnné, imaginární část není omezena.

1219 Poznámky k inverzním hyperbolometrickým funkcím pro komplexní čísla (C99)



I když zde hovoříme pouze o funkcích pro typ `double complex`, platí uvedené poznámky pro funkce pro všechny komplexní typy.

- Funkce `cacosh()`, jež počítá komplexní hyperbolický arkuskosinus, má řez podél reálné osy pro hodnoty menší než 1. Imaginární část argumentu musí ležet v intervalu $\langle -\pi, +\pi \rangle$.
- Funkce `casinh()`, jež počítá komplexní hyperbolický arkussinus, má řez podél imaginární osy v výjimkou intervalu $\langle -i, i \rangle$.

- Funkce `catanh()`, jež počítá komplexní hyperbolický arkustangens, má řez podél reálné osy mimo interval $\langle -1, 1 \rangle$.

1220 Poznámky k dalším funkcím pro komplexní čísla (C99)



I když zde hovoříme pouze o funkcích pro typ `double complex`, platí uvedené poznámky pro funkce pro všechny komplexní typy.

- Funkce `clog()`, jež počítá komplexní přirozený logaritmus, má řez podél záporné části reálné osy. Hodnoty imaginární části argumentu jsou omezeny na interval $\langle -\pi, +\pi \rangle$.
- Funkce `cpow()`, jež počítá komplexní mocninu x^y , má pro první parametr řez podél záporné části reálné osy.
- Funkce `csqrt()`, jež počítá komplexní druhou odmocninu, má řez podél záporné části reálné osy.

1221 Goniometrický tvar komplexního čísla



Komplexní čísla se často vyjadřují v tzv. goniometrickém tvaru pomocí *absolutní hodnoty* (normy, modulu, magnitudy) r a *fázového úhlu* (argumentu) φ . Platí

$$z = r \cdot (\cos \varphi + i \cdot \sin \varphi) = a + i \cdot b.$$

Absolutní hodnotu komplexního čísla určíme pomocí funkce `cabs()`, fázový úhel určíme pomocí funkce `carg()`. Obě tyto funkce očekávají parametr typu `double complex` a vracejí hodnotu typu `double`.

K dispozici jsou samozřejmě i odpovídající funkce pro typy `float complex` a `long double complex`.



Poznámka: Pro *absolutní hodnotu* komplexního čísla se používají také označení *norma*, *modul* nebo *magnituda*. Pro *fázový úhel* se používá také označení *fáze* nebo *argument*.

1222 Komplexní čísla v C++



Standardní knihovna jazyka C++ obsahuje šablonu `complex<>`, jejímž parametrem je číselný typ. Je definována v hlavičkovém souboru `<complex>` a implementuje třídu pro reprezentaci komplexních čísel se složkami typu určeného parametrem šablony. Vedle implementace pro obecný typ T poskytuje standardní knihovna specializace pro všechny tři typy reálných čísel.

O instancích třídy `complex<>` s parametrem T budeme hovořit jako o *komplexních číslech založených na typu T* .

Komplexní čísla – tedy instance třídy `complex<>` – jsou v paměti uložena jako dvojice reálných čísel typu T , jež představují reálnou a imaginární část. (Stejně jako v C99 se i v C++ používá reprezentace komplexních čísel v kartézských souřadnicích.)

1223 Deklarace komplexní proměnné



Třída `complex<>` má konstruktor, kterému lze předat reálnou a imaginární část vytvářené instance; pro oba tyto parametry jsou definovány implicitní hodnoty 0, takže všechny následující deklarace jsou správné:

```
complex<double> cd(3,5); // vytvoří číslo (3, 5)
complex<double> ce(3);   // vytvoří číslo (3, 0)
complex<double> cg = 7;   // vytvoří číslo (7, 0)
complex<double> ch;       // vytvoří číslo (0, 0)
```

1224 Konverze komplexních čísel (C++)



Ve všech třech specializacích šablony `complex<>` jsou definovány jednoparametrické konstruktory, jejichž parametrem je instance třídy `complex<T>` s parametrem zbývajících dvou typů. Tyto konstruktory umožňují vzájemně převádět komplexní čísla založená na jednom reálném typu na komplexní čísla založená na jiném reálném typu.

Tyto konstruktory jsou ovšem explicitní, takže požadované konverze si musíme explicitně předepsat. To znamená, že musíme psát např.

```
x = z + (complex<double>) y;
```

1225 Přiřazování komplexních čísel



Komplexní čísla založená na různých číselných typech lze vzájemně přiřazovat. O to se starají přetížené přiřazovací operátory.

Vedle šablony prostého přiřazovacího operátoru jsou k dispozici i složené přiřazovací operátory.

Komplexní proměnné lze přiřadit i hodnotu reálného typu. V takovém případě vznikne komplexní číslo, které bude mít nulovou imaginární část.

1226 Aritmetické operace pro komplexní čísla (C++)



Pro komplexní čísla jsou v C++ přetíženy operátory obstarávající všechny potřebné základní aritmetické operace – unární plus a minus, sčítání, odčítání, násobení, dělení. Binární operátory jsou přetíženy i pro případ, že jeden operand je reálného typu.

Operace, v nichž použijeme komplexní čísla založená na různých reálných typech, mohou být z hlediska překladače nejednoznačné – překladač nebude vědět, který z přetížených operátorů použít. Proto je v mnoha případech nezbytné explicitní přetypování.

1227 Relace pro komplexní čísla (C++)



Komplexní čísla lze mezi sebou porovnávat pomocí operátorů `==` a `!=`. Dvě komplexní čísla si jsou rovna, rovná-li se reálná část prvního reálné části druhého a zároveň rovná-li se imaginární část prvního imaginární části druhého.

Přetížené operátory `==` a `!=` umožňují porovnávat komplexní čísla s reálnými; reálné číslo se při takovém porovnání chová jako komplexní číslo s nulovou imaginární částí.

1228 Reálná a imaginární část komplexního čísla (C++)



Reálnou, resp. imaginární část komplexního čísla zjistíme pomocí metody `real()`, resp. `imag()`.

Vedle toho máme k dispozici volné funkce `real<>()` a `imag<>()`, které ve skutečnosti pouze volají uvedené metody.

1229 Vstup a výstup komplexních čísel pomocí objektových proudů (C++)



Komplexní čísla lze pomocí operátoru `<<` vložit do výstupního datového proudu. Komplexní číslo vystoupí v souřadnicové reprezentaci, tedy jako dvojice reálných čísel uzavřených do závorek. Napíšeme-li např.

```
complex<double> x(6,7);  
cout << x << endl;
```

vystoupí

(6,7)

Podobně můžeme komplexní čísla číst pomocí operátoru `>>`. Tento operátor očekává opět spouřadnicovou reprezentaci uzavřenou v závorkách, tedy stejný tvar, jaký jsme dostali na výstupu.

1230 Matematické funkce pro komplexní čísla (C++)



Pro komplexní čísla jsou v C++ přetíženy všechny běžné matematické funkce (trigonometrické a cyklometrické funkce, hyperbolické a hyperbolometrické funkce, exponenciální a logaritmické funkce).

Pro tyto funkce platí vše, co jsme řekli v poznámkách o implementaci matematických funkcí pro komplexní čísla v C99 (tipy 1218–1210).

1231 Komplexně sdružené číslo



Funkce `conj<>()` očekává jako parametr komplexní číslo a vrátí číslo komplexně sdružené. (Viz též tip 1214.)

1232 Goniometrická reprezentace komplexního čísla (C++)



Komplexní čísla mohou být vyjádřena také v tzv. goniometrickém tvaru (viz též tip 1221). Potřebujeme-li zjistit absolutní hodnotu, použijeme metodu `abs()`, a chceme-li zjistit fázový úhel, použijeme metodu `arg()`.

1233 Vytvoření komplexního čísla na základě goniometrické reprezentace (C++)



Komplexní čísla mohou být vyjádřena také v tzv. goniometrickém tvaru (viz též tip 1221). Známe-li goniometrický tvar, získáme souřadnicovou reprezentaci pomocí funkce `polar()`, které předáme jako první parametr absolutní hodnotu a jako druhý fázový úhel.