Výuka jazyka C++ pomocí realizace příkladu s využitím třídy pro komplexní čísla formou otázek a odpovědí

(vývojová verze 0.05)

Struktura textu je taková, že úvodní texty a návodné texty jsou psány italikou. Otázky jsou číslovány X.X a jsou modré. Následují odpovědi. Snažte se zobrazit pouze otázku a odpovědět. Až potom si přečtěte odpověď.
Jelikož programování je svým způsobem umění vytvořit celek z dostupných částí, existuje více správných řešení a tedy vaše řešení může být správné, i když se od uvedeného liší.
Text se zaměřuje především na základní použití mechanizmů C++ (na praktickou stránku). Pro získání detailnějšího a komplexnějšího přehledu doporučujeme použít dostupnou literaturu.

Označení (PC) znamená, že by mělo být Programováno v C – a tedy realizováno na výpočetní technice.

Na konci kapitol je souhrnný projekt, který svou obtížností odpovídá v té době probrané látce a má za úkol procvičit tuto látku.

Projekt komplexní čísla, vykreslení grafu

Navrhněte program pro vykreslení frekvenčních charakteristik ze standardního přenosu v komplexní rovině a logaritmických souřadnicích.
Potřebujeme typ komplexní číslo -> zkusíme ho vytvořit.

1 Rozbor úlohy - Funkce pro vykreslení charakteristiky - přenos

Přenos je komplexní funkce komplexní proměnné, F(p) = F(a + jω) = … nebo F(jω) =…. Znamená to tedy, že se vypočte funkce pro měnící se komplexní parametr a výsledky (komplexní čísla) se vykreslí do komplexní roviny, nebo ve formě dvou grafů: amplituda a fáze.

1.1 Napište funkci, která bude mít v čitateli polynom prvního řádu, ve jmenovateli polynom třetího řádu. Koeficienty čitatele budou b, koeficienty jmenovatele a, proměnná polynomu p. Zapište matematicky jako vzorec a v notaci pro MATLAB. F(p) = ?

$$F(p) = \frac{b_1 p + b_0}{a_3 p^3 + a_2 p^2 + a_1 p + a_0}$$

F1 = (b1 * p + b0) / (a3 * p^3 + a2 * p^2 + a1 * p + a0)

Koeficienty a,b jsou v daném případě reálná čísla, parametr p je komplexní. Výsledek F1 je také komplexní.
Z hlediska frekvenční charakteristiky jsou koeficienty a,b konstanty, p = jω je proměnná.

Znak ^ se v MATLABu používá pro mocninu. !! Pozor v jazyce C znamená něco jiného a pro mocninu se používá funkce pow (tj. power=mocnina) !!
Bylo by výhodné, kdyby náš program „uměl“ MATLABovskou notaci.

2 Třída – nový datový typ - Objektově Orientované programování

V jazyce C++ je možné vytvořit nový datový typ, který „umí“ (je schopen) provést stejné operace jako základní datové typy. Novým datovým typem tedy pro nás bude typ pro komplexní čísla.

Název nového typu může být libovolný. Většinou se volí tak, že začíná T (Type/Typ), případně C (Class/třída), S (Struct/Struktura). Pro nás tedy (například) TKomplex.

Překladač umí nový typ přeložit (ve smyslu „volání“, tj. hlavičky) v situacích ve kterých umí přeložit základní typy. Překladač ale neví, co mají dělat (ve smyslu činnosti, tj. tělíčka), pokud ho to nenaučíme (nenapíšeme danou funkci).

Pro práci při tvorbě tříd platí, že pokud to umím se základním datovým typem, musí to jít (po změně názvu typu) i s novým datovým typem (pokud napíšu příslušné tělíčka, aby překladač věděl co dělat).

2.1 Se kterým datovým typem je lepší představy provádět? S typem int nebo double?
Při úvahách o operátorech přihlédněte k zápisu vzorce F1 = (b1 * p + b0) / (a3 * p^3 + a2 * p^2 + a1 * p + a0), který zároveň později povede k požadavku: „aby šlo používat vzorce stejným způsobem jako v MATLABu“.

Pro představu je lepší použít datový typ int, pro který dává smysl použití většího počtu operátorů (například zmíněný operátor ^, který pro double nedává smysl. I když i pro int je jeho činnost značně odlišná od plánované mocniny).

Z hlediska přesnosti by byl vhodnější typ double, ale ani int ani double nestačí k reprezentaci komplexního čísla (takže pro představu jsou rovnocenné).

3 Tvorba třídy – co by měl typ umět – vycházíme z kódu

Při tvorbě třídy můžeme vycházet z toho, co právě potřebujeme. Napíšeme tedy program (nový typ můžeme pro začátek zaměnit typem int, nejlépe pomocí typedef int TKomplex, abychom po změně nemuseli dělat v programu změny typu (mimo typedef). Tento postup je praktický pro okamžitou činnost psaní programu a pro úvodní úvahy. Je ale nepraktický pro budoucnost, protože kvalitní návrh třídy je možné provést jen z komplexního rozboru (ne z omezeného aktuálního výběru) – viz následující kapitola.

Napište funkci FrChar, která bude mít jako vstup koeficienty čitatele, jmenovatele (stejného řádku), řád, a která vypočte hodnoty pro frekvence od 0.01 až 1000, vypočtené hodnoty vrátí pomocí parametru. (???)

3.1 Jaká bude hlavička funkce? Definujte vstupy a výstupy

int FrChar(double aCitatel[], double aJmenovatel[], int aRad, TKomplex *aData, double *aFrekvence, int aPocetFr)

Funkce vrátí počet naalokovaných dat/prvků, nebo kód chyby.

Všimněte si, že díky typedef jsme odlišili „skutečné“ celočíselné hodnoty od hodnot, které se změní na komplexní čísla.

3.2 Jaké bude tělo funkce? Tělo funkce tvořte spíše jako tvorbu algoritmu s tím, že správné hodnoty budou vypočteny později (až se vytvoří datový typ pro komplexní čísla).
Použijte zatím typ int. Pro výpočet výsledku přenosové funkce použijte formát zápisu jako pro MATLAB (i když zatím nedává smysl). (PC)

Vypadá Vaše funkce nějak takhle?

int FrChar(double aCitatel[], double aJmenovatel[], int aRad, TKomplex *aData, double *aFrekvence, int aPocetFr)
{
naalokujte pole pro uložení výsledku
pro každou frekvenci {
    vypočtěte hodnotu čitatele (pro danou frekvenci)
    vypočtěte hodnotu jmenovatele (pro danou frekvenci)
    vypočtěte výslednou hodnotu přenosu (pro danou frekvenci) a výsledek uložte
}
return chyba nebo délka vektoru;
}

3.3 V jazyce C/C++ by kód mohl být například (PC):

```
double pom; int i,j;
for (i=0;i<Počet;++i) {
 TKomplex p = frekvence[i];
   {citatel = jmenovatel = 0; // naše = bude muset mít návratovou hodnotu, aby se chovalo standardně
  for (j=0;j<rad;j++) {citatel += a[j]*p^j; jmenovatel = jmenovatel + = b[j]*p^j ;}
  výsledek[i] = citatel / jmenovatel;
}
}
```

všimněte si řádku TKomplex p = frekvence[i];. Zde bychom potřebovali, aby se frekvence zapsala do imaginární složky. Ukážeme si i jiné způsoby, pro začátek bychom mohli uvažovat, že máme k dispozici ryze komplexní proměnnou i_komplex (předdefinovanou na 0 + 1i) a tedy můžeme zatím psát
TKomplex p; p = frekvence[i] * i_komplex;.
Zároveň zjišťujeme, že budeme kromě operátorů potřebovat i způsob/funkci pro vytvoření komplexního čísla, nastavení hodnot komplexního čísla a pro zjištění hodnot komplexního čísla.

3.4 Které funkce/operátory bude muset třída umět? S jakými datovými typy budou operátory pracovat. (Slovně/teoreticky)

Při úvahách přihlédněte k následujícímu kódu a výsledkům:

```
int i,j; double x,y;
xx = i  /  j; // jakého typu jsou parametry, a jaký je typ výsledku?
xx = x / y; // jakého typu jsou parametry, a jaký je typ výsledku?
xx = i  / x; // jakého typu jsou parametry, a jaký je typ výsledku?
xx = x /  i; // jakého typu jsou parametry, a jaký je typ výsledku?
```

(jedná se vlastně o čtyři operátory dělení, mající různé parametry)

Při rozboru kódu zjistíme, že budeme potřebovat operátory:
double * TKomplex pro násobení koeficientů s komplexním operátorem p, výsledkem by měl být přesnější z typů, tedy TKomplex (stejně jako v následujících případech)
TKomplex^int pro mocnění (porušujeme zde pravidlo, že operátor by měl pokud možno dělat v nové třídě věci stejné (nejhůře pak podobné) jako u standardních tříd – aby byl program čitelný. Na druhou stranu musíme splnit podmínku kompatibility zápisu s MATLABEM, které zde dáme přednost.)
alternativně: TKomplex^double – je univerzálnější než ^int, možná pomalejší (?)
TKomplex + TKomplex (nebo TKomplex +=TKomplex v závislosti na použitých operátorech – zkuste čitatel jedním a jemnovatel druhým operátorem) pro sčítání prvků v čitateli a jmenovateli
TKomplex = TKomplex - pro přirazení výsledku výpočtu na pravé straně (a = b * c, kde výsledkem b *c je TKomplex)
TKomplex / TKomplex (nebo TKomplex/=TKomplex) pro výpočet podílu čitatele a jmenovatele

4 Tvorba třídy – co by měl typ umět – vycházíme z rozboru

Tvorba třídy se skládá z několika úvah (které je nutné řešit současně (nebo v návaznosti) jako celek)

4.1 Jaká data má komplexní číslo (a tedy i třída pro tento typ)? Proveďte diskuzi nad dvojím možným vyjádřením komplexního čísla, který formát je lepší? Jaký standardní datový typ pro tato data použijete?

Komplexní číslo má dvě složky. Existuje složkový zápis (reálná a imaginární část) a exponenciální (Amplituda a fáze). První zápis je vhodnější pro sčítání, druhý pro násobení (mocniny…). Jelikož obecně nevíme, které operátory se budou vyskytovat častěji, můžeme oba zápisy považovat za rovnocenné.

Druhou otázkou je zda používat jeden z formátů (dvě proměnné na typ), nebo oba zaráz (čtyři proměnné na typ). Patrně lepší přístup (z hlediska orientace) je lepší mít dvě proměnné a v případě potřeby přepočítávat, než po každém výpočtu (nezapomenout) dopočítat druhý typ.

Pro reprezentaci dat (složek) je vhodný reálný datový typ – double (pokud nepotřebujeme extrémní přesnost).

U tříd dále mohou existovat pracovní/režijní/pomocné proměnné. Například proměnná index s jednoznačnou identifikací proměnné, nebo proměnná pro uložení stavu proměnné (neinicializovaná/inicializovaná/neplatná (po chybě)/…)

Další částí je vnik a zánik typu. Jedná se o stav, který známe u standardních typů jako definici s inicializací. Zánik standardních typů realizuje překladač a programátora se netýká.

4.2 Co se děje při následujících definicích?
int aa, bb = 3, cc = bb, dd = 8.5;

Pro prvek aa se pouze vytvoří paměť na uložení, inicializace konkrétní hodnotou se neprovede. Jedná se o nejjednodušší vytvoření prvku, které se používá, není-li uveden parametr – proto se nazývá implicitní konstrukce.
Paměť rezervovaná pro prvek bb naplní hodnotou 3. Z hlediska mechanizmu překladu znak '=' neznamená přiřazení ale inicializaci při vytvoření prvku (konstrukci hodnoty nového prvku na základě hodnoty). Jelikož se jedná o přiřazení int hodnoty do typu int, vytváří se kopie. Jedná se tedy o konstrukci kopírovací.
Paměť prvku cc se naplní hodnotou v prvku bb. Jelikož oba prvky jsou typu int, jedná se o předchozí případ, kdy se vytváří kopie.
Paměť prvku dd se naplní hodnotou 8.5 (?). Hodnota 8.5 je ovšem typu double. Proto musí dojít ke změně = konverzi na cílový typ, tj. typ int. Proto se inicializace (každá inicializace s jedním parametrem jiného typu) nazývá konverzní.

U nových datových typů můžeme činnost i typ a počet parametrů nadefinovat. Znamená to, že i implicitní konstrukce může provádět inicializační činnost (na rozdíl od standardních typů). Dále můžeme vytvářet inicializaci s více hodnotami. Proto je zavedena (i pro standardní typy) možnost inicializaci zapisovat jako funkci, ve tvaru proměnná a v závorce parametry, ze kterých bude vytvořena. Pro náš příklad je možné psát i: int aa, bb(3), cc(bb), dd(9.5);

4.3 Jaké způsoby inicializace byste navrhli pro typ TKomplex a co by dělaly? (Zatím pouze ekvivalenty inicializace z minulého zadání).
Nejprve popište slovně bez návaznosti na jazyk C++.
Uveďte zápis definice proměnných s inicializacemi, které navrhnete. Nejprve použijte předchozí definici pro int, kde zaměníte typ na TKomplex a popište, jaká by byla činnost.

Slovně (všimněte si, že tato část přesně popisuje, co se stane, ale vychází pouze ze znalosti komplexních čísel a formulací činností. Neříká jak se bude realizovat v programu.): Proměnná pro komplexní číslo se bude vytvářet bez dodaných dat – výsledkem bude komplexní číslo s nulovými hodnotami. Proměnná se bude vytvářet jako kopie jiného komplexního čísla. Proměnná komplex se bude vytvářet z jiného základního datového typu (hodnotového, mimo ukazatelů …) tak, že se tato hodnota stane reálnou složkou, imaginární složka bude nulová.
TKomplex aa, bb(3),cc(bb),dd(9.5);
Zde se odrazíme od pravidla, že co funguje pro int, musí fungovat pro náš typ. Zbývá „pouze" říci co se bude dít.
Pro aa se vytvoří „implicitní", „prázdný", … prvek. Názor na to, co to je, a jak to vypadá, se může lišit. My zvolíme možnost, že vznikne komplexní číslo s nulovými složkami. Všimněme si, že navrhujeme implicitní

inicializaci, která u standardních typů nic nedělá, ale u nových typů je možné ji vytvořit/předepsat.
Pro bb(3) se jedná o konverzi z typu int. Komplexní číslo má ale dva parametry. Opět můžeme realizovat prakticky cokoli nás napadne, a dokážeme to naprogramovat. Ale nejrozumnější patrně bude přiřadit hodnotu do reálné složky (což se děje velice často) a složku imaginární vynulovat.
Pro dd(9.5) se opět jedná o konverzi, tentokrát z typu double. Otázkou je, zda potřebujeme konverzi z int i z double, které dělají totéž. Výsledkem úvahy by asi bylo, že konverzi z int nepotřebujeme, protože při volání s typem int ho dokáže překladač konvertovat na double a zavolat inicializaci s typem double.
Pro cc(bb) se jedná o vytvoření kopie. Činnost je jasná – reálná složka nové proměnné cc bude kopií reálné složky původní proměnné bb, a podobně pro složky imaginární.

4.4 Jaké další způsoby inicializace byste navrhli pro typ TKomplex a co by dělaly? Uveďte textový popis, zápis definice proměnných s inicializacemi, které navrhnete.

Obecně slovně: Proměnná typu TKomplex půjde vytvořit na základě řetězce ve formátu …, dále půjde vytvořit ze dvou hodnot, kde první hodnota bude reálná složka. Proměnná půjde také vytvořit tak, že načte data z otevřeného souboru (kde budou uložena ve stejném formátu jako pro inicializaci z řetězce).
TKomplex ee(“10+56i“),ff(“<12;13.5>“), gg(5,8.3), hh(6.2,7,0), mm(otevreny_soubor);
Z řetězce (může být i více typů formátů řetězců, pokud je uvnitř jediné funkce rozpoznáme a zpracujeme)
Z reálné a imaginární složky.
Z amplitudy a fáze. Nutno odlišit od předchozího - typem to rozumně nelze, takže počtem parametrů (poslední parametr slouží k odlišení, ale jeho hodnota se nevyužije).
Načtení hodnot z otevřeného souboru.
a další …

4.5 Stejně jako vznik proměnné je možné ovlivnit i zánik proměnné – k tomu slouží funkce nazývaná destruktor, která řekne, co se má dít při zániku proměnné. Vyjmenujte činnosti, které je nutné udělat před zánikem prvku (nápověda: soubory, paměť, vypočtená data …).

Jelikož se zánikem prvku „zmizí“ i jeho data, je nutné ošetřit situace, kdy by tato ztráta mohla být „nepříjemná“. To jsou například následující situace:
- zmizí vypočtená data – je nutné je nejprve uchovat/uložit,
- zmizí unikátní odkaz na získané zdroje (například ukazatel na alokovanou paměť – nutno odalokovat, odkaz na otevřený soubor – nutné uzavřít, …)
Při „záchraně“ dat je ale nutné uvažovat o vlivu na obsluhu – „grafický“ dotaz na uložení dat nemůžeme použít u typu, jehož proměnných jsou v programu vytvořeny desítky a více (komplexní číslo, řetězec, matice …). Toto je možné pro jednu či dvě hodnoty (například dokument word, excel …)

Pro datové typy s více proměnnými je nutné navrhnout i zadávání a čtení hodnot. Při zadávání je nutné uvažovat i způsoby kontroly správnosti dat (například, že délka nemůže být záporná …).
Stejně je nutné vyřešit i čtení hodnot. Zadávané/čtené hodnoty nemusí být přístupné přímo, ale mohou být

přepočítávány (například z více „vnitřních“ hodnot bude jedna výsledná). Může také existovat více výstupních reprezentací (pohledů) na data.

4.6 Popište/navrhněte způsoby nastavování a čtení dat pro typ TKomplex.

Nastavování je vhodné uvažovat pro jednotlivé složky a jako celek – odpovídaly by tomu názvy funkcí SetReal, SetImag, Set (=SetKomplex). Otázkou je, jak by se první dvě funkce zachovaly ke „druhé“ složce (zda by ji nulovaly, nebo ji ponechaly nezměněnou). Podobný problém vyvstane zřetelněji při druhém způsobu, kdy dává smysl patrně pouze společné zadání amplitudy a fáze SetAF a následný přepočet na vnitřní data reálné a imaginární složky.
Můžeme také vytvořit proměnnou z řetězce Set(“10+5i“). Funkce se může jmenovat stejně jako jiná, pokud má jiné parametry (zde má funkce set jeden parametr char*, a tedy se výrazně liší od Set uvedené výše, která má dva parametry typu double).
Pro čtení by bylo vhodné volit GetReal, GetImag. Také pro druhou reprezentaci GetAmpl, GetFaze. Představené by mohly vracet přímo dotazované hodnoty. Pokud by byl požadavek na získání obou parametrů zaráz, musely by být vráceny pomocí dvou parametrů funkce (např.) GetRI, GetAF.

V případě implementace servisní nebo chybové proměnné by bylo nutné uvažovat i o manipulaci s těmito proměnnými.

4.7 Navrhněte funkce pro práci s komplexními čísly. (Mimo operátory zmíněné dříve)

Abs
Doplněk
Inverze
Mocnina
Porovnávání podle různých kritérií (Problémem je, že jsou zde různé možnosti zápisu a tedy více hodnot, ale výsledek je jen jeden. Jedno, základní porovnání, by bylo vhodné zvolit pro operátory ==, <.> …)
a mnoho dalších

4.8 Hodnoty je také vhodné ukládat (zobrazovat) a načítat – navrhněte vhodný textový formát pro typ TKomplex.

Možné formáty již byly uvedeny dříve při inicializaci. Je možné volit libovolný formát. Formátů je možné volit i více (na úkor přehlednosti a jednoznačnosti čtení a složitosti obslužných funkcí). Obecně je vhodné, aby byl program schopen načíst řetězec, který vytiskne. Při tvorbě je důležité vhodně zvolit oddělovače (například čárka jako oddělovač může být zaměněna s desetinnou čárkou (při „českém“ znakovém prostředí), neoddělené proměnné mohou splývat při ukládání polí – srovnejte orientaci v poli komplexních čísel:
15;32;35.5;123;54.77;1;
<15;32><35.5;123><54.77;1>

4.9 Díky novým možnostem bylo možné realizovat jednotný způsob vstupu a výstupu, Proto je nutné vybrat jeden z možných formátů jako prioritní a napojit ho na tento mechanizmus.

Pro tento mechanizmus jsou použity operátory bitového posuvu << a >>.

4.10 Projekt: Navrhněte (slovně popište) požadavky na vytvoření třídy pro dvourozměrné pole. Požadavky popište slovně (laicky, bez nutnosti znalosti jazyka. Napište, co by se Vám líbilo aby typ Pole2D uměl). Návrh napište v kategorií: „proměnnou typu Pole2D by bylo potřeba vytvořit: bez parametrů, v případě jednoho parametru typu … bude tento znamenat …, v případě jedné proměnné typu char ukazatel bude v této proměnné řetězec pro tvorbu proměnné ve formátu … … … . Při zániku proměnné ne/chci uložit data. Hodnoty v proměnné chci číst, … Práce s proměnnou budou realizovány funkcemi …, které dělají … Pole bude umět operátory: =, +, +=, … > … , které budou dělat tyto činnosti …

…

Pozn.:

Zadání by nemělo jít příliš "hluboko", řešením vnitřní reprezentace (proměnných). To není na škodu, pokud je uvedeno, ale vede to k tomu, že se v tom motá více pohledů a celkově je pak zadání slabé.

Různé formy zadání by bylo vhodné od sebe oddělit (obecné zadání popisující pouze činnosti; zadání rozšířené o datové typy členských dat a datové typy parametrů funkcí; konkrétní realizace)

Zadání by se mělo napsat ve stylu textu "co po té třídě (času, datumu, zlomku…) budeme chtít, co by se s tím dalo dělat" bez návaznosti na jazyk C/C++.

Napsat tedy "hodnota bude v intervalu od ... do ... a to pouze celá (nebo desetinná, nebo řetězec) čísla" a při další fázi (až se změníte na programátora) teprve uvažovat o řešení.

Dále slovně uvést jak proměnná vzniká (a jak se dodaný údej reprezentuje) - tj. bez údaje, s jedním/2/3/.../100 údaji a jak se z těch údajů vytvoří výsledek, když jsou těmi údaji čísla, znaky ... (například jak se z jedné hodnoty typu int vytvoří čas (jsou to např. sekundy), zlomek (je to celá část), interval (je to vyšší mez, nižší je nula) ...)

Dále nepište to ve stylu "... může ... , ... nebo ..., ". Teď vytváříte zadání, které musí být jednoznačné - jinak se to podle něj nebude dát naprogramovat.

Dále nezapomínat na operátory - například řeknu, že chci hodnoty sčítat (a musím napsat, jak se podle mě součet chová - ale opět slovy - například "je-li součet větší než 100 (nebo 200), upraví se výsledek tak aby byl v intervalu 0-100 (0-200)".

U odečítání například "je-li výsledek záporný změní se na kladný tak, že..."

Zkrátka dáváte programátorovi pokyny, co to má dělat a jak si to představujete - vlastní realizaci pomocí jazyka zatím neřešte.

Vypracujte samostatně. V dalších částech naprogramujte a zhodnoťte, zda vaše představy byly v pořádku. Nedostatky plynoucí z nedostatečného pochopení návrhu v této fázi učení opravte.