Objektově orientované programování

Martin Pergel

11. října 2015

Pragmatické informace

- Volitelný/povinný předmět,
- zápočet: zápočtový program (s dokumentací), aktivní účast na cvičení (body v CodExu), praktický test,
- zkouška: objektový návrh, otázky.
- Podrobnosti možná s kolegou Holanem pozměníme!

Cíle předmětu

- Programování v C, C++ a C#,
- objektové programování,
- moderní vývojové prostředky.

Souvislosti

- Prvácké Programování II s kódem NPRG je výrazně odlišné od Programování 2 s kódem NPRM, resp. NMIN,
- proto matematici nejsou schopni navštěvovat prakticky žádné navazující přednášky o programování.
- Kolegové Matematici požádali o C++, ale bez C# nelze navštěvovat informatický druhácký C#.
- Budeme masívně stavět na Pascalu.
- Uvidíme, že všechny jazyky jsou vlastně stejné (při troše štěstí).

Prostředky

- Budeme používat CodEx,
- využívat budeme buďto Microsoft Visual Studio (v dostupné verzi),
- nebo prostředí MONO (obojí je k dispozici legálně zdarma aspoň v omezené verzi).
- Tvořit budeme převážně konzolové aplikace (o psaní formulářových si taktéž něco řekneme - především v C#).

Literatura

- Bruce Eckel: Thinking in C++
- Bruce Eckel: Myslíme v jazyku C++ (Grada)
- Miroslav Virius: Pasti a propasti jazyka C++
- Andrew Koenig, Barbara E. Moo: Accelerated C++
- Stanley B. Lippman: Essential C++

Historie

- Navržen v Bell Labs v rámci vývoje UNIXu,
- podle K. Thompsona dostali B. Kernighan a D. Ritchie uloženo vytvořit obfuscovaný Pascal,
- navrhli jazyk A, se kterým nebyli spokojeni,
- proto navrhli jayzk B, ...,
- spokojeni byli až s jazykem C,
- jazyk se ovšem začal masívně využívat.

Všeobecné vlasntosti jazyka C

- Syntax vychází z Pascalu,
- plno věcí je ale navrženo odlišně,
- jazyk povoluje věci v Pascalu nemyslitelné (konverze),
- jazyk je výrazně zjednodušen (není string a boolean).
- Existuje několik norem (K & R, ANSI C89, C99, C11).
- Plíživě dochází k omezování možností, C++ možnosti omezuje ještě více.

C VS C++

- C je jazyk podobný Pascalu,
- C++ je jazyk, ve kterém je k vidění téměř vše,
- mají velký průnik syntaxe společný, C++ podporuje objekty, C podporuje kouzlení.
- Obvykle uvidíte překladače C++, ale nikdy nevíte, kdy budete potřebovat C, proto si řekneme o obou jazycích.
- Překladače: GCC, G++, Visual Studio .NET,...

Datové typy a proměnné

- Ve všech jazycích pracujeme s proměnnými,
- datové typy se podobají Pascalu, jen o nich víme méně (a jmenují se jinak).
- `char, byte, int, float, double`
- modifikátory (intu): `long, short, unsigned`.
  Například: `unsigned long int`.
- Ovšem stačí jen: `short` nebo `unsigned long`.
- Definice proměnné: `var a,b,c:integer;` ⇒ `int a,b,c;`
- modifikátory `const` a `volatile`.
- Nechybějí ve výčtu některé důležité datové typy?

Chybějící datové typy

a jak se s nimi vypořádáme:

- `boolean` nahrazen `intem`,
- nula je `false`, cokoliv jiného znamená `true`.
- Místo `stringu` použijeme pointer na `char`,
- jednotlivé prvky stringu budeme ukládat na adresy po sobě jdoucí, pointer ukazuje na začátek.
- V C++ tyto typy již jsou, nicméně procvičení pointerů není nikdy od věci.
- Než se dostaneme k práci s pointery, není moc co cvičit.

Od Pascalu k C

v Pascalu umíme především:

- Tělo hlavního programu,
- používat bloky (begin, end),
- komentovat zdrojové texty,
- definovat funkce a procedury,
- používat (definovat a inicializovat) proměnné,
- vracet návratovou hodnotu,
- pracovat s operátory,
- používat základní řídící struktury.

Case sensitivita

je stejná skoro u všech jazyků z familie C

- Oproti Pascalu je C, C++, C#, Java, Javascript... case-sensitive,
- tedy záleží na velikosti písmen.
- Funkce `writeln`, `Writeln`, `WriteLn`, `writeLn` a `WRITELN` by byly formálně různé funkce (funkce s různými jmény).
- Ovšem pozor, i funkce (pro vstup a výstup) se v různých jazycích jmenují různě!
- Je tudíž třeba učit se jména funkcí i s velikostí písmen.
- Modernější vývojová prostředí našeptávají, je ale dobré nestat se na těchto prostředích naprosto závislým.

Hlavní program

- V Pascalu je hlavní program volně sypaný mezi `begin` a end.
- Jazyk C je metodičtější: funkce `main`,
- tato funkce bere dva argumenty: Počet argumentů (int) a dotyčné argumenty (pole stringů, tedy char-pointerů). Nikdo ale nekontroluje, zda hlavička odpovídá (tedy i `int main()` nebo `int main(void)` bude fungovat.
- Vrací int (čímž může říct, zda se program nedostal do potíží podobně jako v Pascalu).
- Občas se vstupní bod programu jmenuje jinak (`_tmain`) nebo (`wmain`).
- Tyto funkce mohou umožnit pohodlnější práci například s parametry (pokud mají smysl).

Bloky a komentáře

- V Pascalu se bloky zahajovaly a ukončovaly slovy `begin` resp. `end`.
- V jazyku C se zahajují resp. ukončují složenými závorkami (které v Pascalu sloužily k psaní komentářů).
- Tudíž se i komentáře musejí v C syntakticky lišit. Jsou zde dvě možnosti:
- Komentář obecný: `/* Zde je komentar... a bude az do ukonceni komentare jako v Pascalu. */`
- Komentář jednořádkový (pozor, proti ANSI C89):
  `// Komentar do konce radku.`

Procedury a funkce

aneb proč jsem už v prváku nerad slyšel o procedurách

- Procedura je součást programu, která umí načíst a zpracovat zadané parametry.
- Funkce je součást programu, která umí načíst a zpracovat zadané parametry a vrátit výsledek.
- Vidíme, že procedura je jen funkce, která nevrací hodnotu, proto v C procedury nejsou.
- Procedura je funkce vracející nic. Syntakticky tedy zavedeme prázdný datový typ `void`.

Procedury a funkce

příklad

- V Pascalu: `function nazev(arg1:typ1; arg2:typ2;...):navr_typ;`
- V C: `navr_typ nazev(typ1 arg1, typ2 arg2,...)`
- Poznámky: Typ parametru musíme uvést pokaždé zvlášť (nefunguje pascalské `a,b,c:integer`).
- Povšimněte si středníků na konci hlavičky. Uvedeme-li středník v C, jde o deklaraci (za kterou nebude následovat definice - ekvivalent pascalské direktivy `forward`).
- Funkce se volají téměř jako v Pascalu, jen je třeba vždy použít operátor zavolání (kulaté závorky s argumenty – byť prázdnými)!
- Tři tečky lze takto v C opravdu použít (ale o tom později).

Definice proměnných

a inicializace – definice je skoro jako v Pascalu, inicializace je úplně jiná

- V Pascalu: `var a,b,c:integer;`
- V C: `int a,b,c;`
- V Pascalu byl problém s inicializací,
- V C není: `int a=0,b=1,c=2;`
- Podle C86 bylo třeba lokální proměnné definovat na začátku funkce, podle C99 lze prakticky kdekoliv (ale není vhodné toho zneužívat).
- Definice globálních proměnných je volně sypaná ve zdrojáku (zdroják sestává především z definic proměnných a funkcí).

Návratová hodnota

funguje úplně jinak než v Pascalu, proto pozor

- V Pascalu byla speciální write-only proměnná, do které se návratová hodnota přiřazovala.
- Jazyk C vychází z toho, že cílem funkce je vrátit návratovou hodnotu, tedy
- jak je návratová hodnota jasná, funkce končí.
- Návratovou hodnotu vrací `return...`
- Příklady: `return;`, `return hodnota;`

Příklad

využívající operátory, o kterých si něco řekneme za chvíli...

V Pascalu:

```
var globpna:longint;{Globalni promenna}
function secti(a,b:integer):integer;
var pom:integer;{lokalni promenna}
begin
     pom:=a+b; {Secteme}
     secti:=pom; {Vratime hodnotu}
end;
```

Kdežto v C: `long globpna; //Globalni promenna`

```
int secti(int a, int b)
{    int pom=a+b; //Secteme pri inicializaci
     return pom; //Slo by i return a+b;
     /* Pozor, instrukce zde uz neprobehnou! */
}
```

Základní operátory

nejsou uvedeny podle priorit!

- aritmetické: $+, -, *, /, \%$ (součet, rozdíl, součin, podíl, zbytek po dělení – pozor, podíl sleduje typy argumentů),
- logické: $\&\&, ||, !$ (konjunkce, disjunkce, negace)
- relační: $<, >, >=, <=, ==, != $ (menší než, větší než, větší nebo rovno, menší nebo rovno, rovno, nerovno)
- přiřazovací: $=, +=, -=, |= \ldots$ (přiřaď, přičti, odečti, vyoruj...)
- Pozor, přiřazovací operátory vracejí hodnotu. Návratová hodnota je přiřazovaná hodnota a jsou asociativní zprava.
- Důsledek: `a=b=c=1;` `a=(b=(c=1)+1);`
- Bitové: $\&, |, \hat{}, <<, >>$ (bitové and, or, xor, shift to left, shift to right např. `(16>>2) == 4`)
- Inkrementace, dekrementace: $++, --$. Pozor na side-efekt!
- A pozor na priority!

Poznámka

jejíž důležitost se projeví za chvíli

Zatímco v Pascalu se středníkem statementy oddělovaly, v jazycích z rodiny C se středníkem statementy ukončují.

Základní řídící struktury `if`

- Podmínky se reprezentují podobně jako v Pascalu, tedy klíčovými slovy `if` a `else`. Syntax se trochu liší.
- `if(podm) prikaz_nebo_blok`
- `if(podm) prikaz_nebo_blok else prikaz_nebo_blok`
- Příklad: `if(teplota>25) printf("Jdu do hostince\n");`
- `if(teplota>25) printf("Do hostince\n"); else printf("Nikam!");`
- `if(teplota>25) { teplota=teplota-25; printf("Putyka vola!\n");}`
  `else printf("Nic nebude!\n");`
- Pozn: String uzavíráme do uvozovek, znak (typ `char`) do apostrofů.

Pozor!

Podmínky v jazyce C jsou záludné.

- V jayzku C není typ `boolean`. Podmínka se reprezentuje `intem`.
- `if(0)....` vždy nesplněná podmínka
- `if(11)....` vždy splněná podmínka
- `if(11 && 1)...` vždy splněná podmínka
- `if(a=1)...` vždy splněná podmínka

Warningy a errory

- Errory známe z Pascalu (znamenají, že jsme něco pokazili),
- jazyk C stanoví, že překladač může varovat (například před jazykovou spodobou)
- na `if(a=1)` obvykle přijde warning (že možná nevíme, co děláme).
- warningu nás obvykle zbaví `if((a=1))`
- Errory jsou předepsány normou, warningy může překladač vydávat dle uvážení.
- Je vhodné kód ladit, dokud není bez warningů.

`while`

základní řídící struktury 2

- Ovládá se analogicky, tedy
- `while(podm) prikaz_nebo_blok`
- Příklad: `while(teplota-->25)printf("Pivo!\n");`
- Pozor, pokud cyklus neproběhne, proměnná teplota se sníží o jedna (přestože tělo neproběhne)!

Náhražka `repeat ... until`

je v podobě konstrukce `while`

- do ... while(podm);
- Opakujeme, dokud je podmínka splněna (oproti Pascalu, kde opakujeme, dokud je podmínka nesplněna).

`for`-cyklus

je v C výrazně výkonnější nežli v Pascalu

- `for(init;podm;increm)prikaz_nebo_blok`
- Příklady: Pascal: `for i:=1 to n do neco`
  C: `for(i=1;i<n;i++) neco`
- Cokoliv může být prázdné (init, increm, tělo i podmínka).
- Prázdná podmínka vždy platí.

Faktoriál

už raději jen bez rekurze...

```
int faktorial(int a)
{    int fakt;
     for(fakt=1;a>1;a--)
          fakt*=a;
     return fakt;
}
```

Faktoriál podruhé

opět raději bez rekurze...

```
int faktorial(int a)
{     int fakt=1;
      for(;a>1;fakt*=a--);
      return fakt;
}
```

Faktoriál potřetí

opět bez rekurze a ještě zvrhleji...

```
int faktorial(int a)
{     int fakt=1;
      for(a++;--a;)fakt*=a;
      return fakt;
}
```

Prázdný `for`-cyklus

umí být nečekaně účinný...

```
for(;;);
```

switch

jak se asi udělá pascalské `case ... of ...`?

```
switch(vyraz)
{     case prvni: kod;
      break;
      case druhy: dalsi kod;
      break;
      case treti: jeste;
            dalsi;
            kod;
      break;
      default: co udelat jinak;
}
```

Slovo `break`

není povinné a jaképak jsou z toho důsledky...

- Neuvedeme-li na konci bloku slovo `break`, kód pokračuje dalším blokem!
- Příklad:

```
switch(den_v_tydnu)
{    case 7: den_v_tydnu-=7;
     case 0: printf("Pondeli\n");
     break;
     case 1: printf("Utery\n");
     break;
     ...
}
```

Header-fily I

čili hlavičkové soubory

- C je versatilní, tudíž do něj nejsou zadrátované skoro žádné funkce. My knihovní funkce používat chceme, a je to možné.
- Podle K& R-normy se nic nekontrolovalo, zavolání neexistující funkce zjistil až linker. Špatné parametry nikdo nekontroloval.
- Podle novějších norem je možné (a vhodné) kontrolu podstoupit.
- Deklarace funkce – hlavička ukončená středníkem (namísto pascalského `forward`).
- Tyto deklarace (spolu s konstantami resp. makry) jsou v hlavičkových souborech `.h`.
- Chceme-li nějakou funkci použít, je vhodné (v C++ nutné) hlavičkový soubor naincludovat.
- `#include<stdio.h>` – soubor pro standardní vstup a výstup.

Header-fily I

čili hlavičkové soubory

- Uděláme-li si vlastní soubor v současném adresáři, includujeme s uvozovkami (místo se zobáčky).

Stručně o vstupu a výstupu

ať můžeme cvičit v CodExu

- funkce `int getchar();` vrací znak nebo údaj o chybě.
- Konec vstupu označí konstantou `EOF`.
- Jak `getchar` tak `EOF` jsou v `stdio.h`.
- Funkce `int putchar(int);` vypíše znak.
- Vypisuje se pomocí `int printf(char format[],...);`
- `printf("Sbohem svete!\n");`
- `printf("%d",cislo);` – vypis cisla.
- Vše je v `stdio.h`.
- Podrobnosti budou brzy.

Příklad

```
#include<stdio.h>
int main(int argc, char*argv[])
{    printf("Sbohem svete!\n");
     return 0;
}
```

Příklad

další

```
#include<stdio.h>
void main(void)
{    int znak;
     printf("Zadej (1cif) cislo: ");
     znak=getchar();
     if(znak==EOF)
     {    printf("Ale, ale!\n");
          return;}
     if((znak<48)||(znak>57))
     {    printf("Co to je!?\n");
          return;}
     znak-=48;
     printf("Napsal jsi %d\n",znak);
}
```