Vyšší programovací jazyky

Programovací jazyky se dělí na *nižší* (např. assembler neboli jazyk symbolických instrukcí) a *vyšší*. Vyšší programovací jazyk je formální jazyk, ve kterém se algoritmus zapisuje strukturovaně. Zápis je srozumitelný, připomíná běžný jazyk (anglický) a matematický zápis; jazyky nejsou závislé na strojových principech počítače (procesoru, architektuře).

Vyšší programovací jazyky jsou dvojího typu: *imperativní* (příkazové) a *neimperativní*.

Program zapsaný v *imperativním* jazyku (např. Cobol, Pascal, Ada, C/C++, Basic, Java, Javascript, php, Python) je založen na zápisu algoritmu posloupností příkazů (zahrnující také cykly a větvení), pro práci s daty se využívají proměnné.
V neimperativním jazyku je program zpravidla tvořen množinou nějakých odvozovacích pravidel, např. logických formulí. *Neimperativní jazyky* se dělí na *funkcionální* a *logické*.

Mezi funkcionální jazyky patří např. Lisp (jehož varianta AutoLISP je součástí AutoCADu). Program ve funkcionálním jazyku je vlastně množina (ne posloupnost!) funkcí, většinou neexistují proměnné, místo nich se pracuje se seznamy dat.

V *logickém jazyku* je program zpravidla tvořen množinou nějakých odvozovacích pravidel, např. logických formulí. Logické jazyky se využívají zejména v umělé inteligenci, v expertních systémech. Typickým představitelem logického jazyka může být Prolog (logický jazyka vestavěný do databází a odvozený od Prologu je Datalog).
Ukážeme si zápis programu v Prologu jako množinu pravidel, který umí řešit problémy vztahů mezi členy rodiny.

Definujeme klauzule:

```
rodic(Jana,Petr) % Jana je rodič Petra
rodic(Petr,Eva)  % Petr je rodič Evy
muz(Petr)        % definuji, že Petr je muž
zena(Jana)       % Jana je žena
```

Definujeme odvozovací pravidla:

```
syn(Y,X) :- rodic(X,Y), muz(Y). % čárka znamená logickou spojku „a“
dcera(Y,X) :- rodic(X,Y), zena(Y).
prarodic(X,Z) :- rodic(X,Y), rodic(Y,Z).
potomek(Y,X) :- syn(Y,X);dcera(Y,X). % čárka znamená logickou spojku „nebo“
```

Pak můžeme klást dotazy:

```
?-rodic(A,Petr)
?-prarodic(Jana,Eva)
```

Systém odpoví na první dotaz `A=Jana` a na druhý `yes`.

V Prologu lze psát i výpočty, např. výpočet faktoriálu rekurzí.

Poznámka: Určitou skupinou jazyků imperativních jsou také *objektově orientované jazyky* (např. C++, Java), objektové rysy byly přidány i do jiných jazyků (php od verze 5, Python).

Náplní tohoto předmětu je výuka „klasického“ jazyka C se zaměřením na speciality a detaily jazyka.

Syntaxe a sémantika

S programovacími jazyky souvisejí dva pojmy, které je nutné znát, a to *syntaxe* a *sémantika*. Syntaxe definuje (zjednodušeně) pravidla zápisu programu (obecně v jazycích větnou skladbu), říká např., kde je v zápisu programu závorka, že každý příkaz je ukončen středníkem apod. Syntaxe jazyka je definována gramatikou, často se čtenář může setkat se syntaktickými diagramy. Sémantika pak určuje význam jednotlivých vět (neboli význam zápisu, který pomocí syntaktických pravidel vytvořen).
Příklad: Syntaxe jazyka C říká, že cyklus s podmínkou na začátku se zapisuje:
`while (podmínka) příkaz;`
Sémantika (význam) takto zapsaného cyklu je: příkaz se provádí opakovaně tak dlouho, *zatímco je* podmínka splněna.

Kompilační a interpretované jazyky

Textový zápis programu ve vyšším programovacím jazyce se nazývá *zdrojový kód*. V případě *kompilačního* programovacího jazyka je zdrojový kód vstupem speciálního programu, který se nazývá *překladač* (kompilátor, compiler). Překladač provede nejprve syntaktickou kontrolu celého textu (kontrolu, zda je program zapsán podle pravidel jazyka); pokud se vyskytne chyba v zápise programu, překlad je přerušen a seznam chyb je zobrazen uživateli. Program bez syntaktických chyb je přeložen do strojového kódu (a optimalizován) do formy samostatně spustitelného souboru (soubory .exe nebo .com v operačních systémech firmy Microsoft, ELF formát v systému Linux). Ten je pak možné spustit samostatně. Zástupci kompilačních jazyků jsou např. Pascal, C/C++.

U interpretovaných jazyků není výsledkem procesu zpracování zdrojového kódu samostatně spustitelný soubor. Abychom program spustili, musíme mít program, který se nazývá *interpret*. Vstupem interpretu je také zdrojový kód. Rozdíl je, že interpret přečte jeden řádek zdrojového kódu, provede syntaktickou kontrolu a ihned zajistí jeho provedení (vykonání). Na chyby v zápise programu se tedy přijde až při běhu programu (běh programu se v tomto případě přeruší). Příkladem interpretovaných jazyků je Basic, Python, JavaScript, Visual Basic Script.

Každý z obou přístupů má své výhody a nevýhody. Výhoda kompilačního jazyka je, že se provede syntaktická kontrola celého textu, výsledný program je možné spustit samostatně bez nutnosti mít speciální programové prostředí. Běžící samostatný program je rychlejší než v případě interpretovaného jazyka. Nevýhodou

je, že při jakékoliv změně programu (zdrojového kódu) je nutné provést znovu překlad. Pokud je potřebné mít program pro různé operační systémy, event. pro různé počítače, je nutné mít překladač pro každý systém.
Hlavní nevýhodou programů zapsaných pomocí interpretovaného jazyka je jejich pomalý běh. Pokud se změní zdrojový kód, není potřeba jej překládat, nový program se spustí pomocí interpretu. Pokud existují interprety pro různé operační systémy, pak je snadná distribuce nových verzí programů uživatelům.

Několik moderních programovacích jazyků spojuje oba principy. Zdrojový kód syntakticky kontroluje překladač a provádí překlad jako u kompilačních jazyků. Výsledkem není samostatně spustitelný soubor, ale speciální soubor obsahující jakýsi „mezijazyk“ (vnitřní formu, virtuální strojový kód). Tato vnitřní forma je pak interpretována interpretem (kterému se také říká virtuální počítač či virtuální stroj). Výhoda tohoto přístupu je zřejmá - interpretace vnitřní formy je rychlejší než v případě klasického intepretačního jazyka, protože je při překladu provedena kompletní syntaktická kontrola a vnitřní forma může být optimalizována. Intepretace je samozřejmě ale pomalejší než při běhu samostatného kódu. Výhodou přístupu je, že mezijazyk může být intepretován stejně na různých platformách, pokud je k dispozici příslušný interpret.
Typickým zástupcem této technologie je jazyk Java. Zdrojový kód (soubory s příponou .java) jsou přeloženy do souboru s příponou .class (což je zmíněná vnitřní forma, v Javě nazývaná bytecode). Bytecode je pak interpretován programem, který se nazývá JVM (Java Virtual Machine). JVM je součástí internetových prohlížečů a interpretuje tzv. applety vložené do webových stránek.
Firma Microsoft představila vývojový systém .NET (Dot Net), který implementuje tento princip. Zdrojové kódy mohou být v zapsány v jazyce Visual Basic NET, Visual C NET, C# (C Sharp) NET. Vnitřní formu nazývá firma Microsoft zkratkou MSIL (Microsoft Intermediate Language), interpret této vnitřní formy se nazývá CLR (Common Language RunTime).

Prezentaci věnovanou vyšším a nižším programovacím jazykům s ukázkami, která byla promítána v rámci předmětu Programování na cvičení 2, si můžete prohlédnout: Programovací jazyky.

Ukázku programu v interpretovaném jazyce Javascript, který je prováděn prohlížečem webových stránek, si můžete také stáhnout (je nutné uložit webovou stránku i soubor s kódem obsah.js a otevírat soubor vypocet_obsahu.html): vypocet_obsahu.html, obsah.js

Poznámka k překladu:
Každý programový balík překladače obsahuje soubory s programovými moduly nazývané *knihovny* (libraries). Jedná se nejčastěji o soubory s příponami .lib nebo .obj, resp. .o. Názorným příkladem knihovny je matematická knihovna. Procesory běžně nemají instrukci pro výpočet např. hodnoty funkce *sin*, logaritmu, odmocniny aj. Hodnoty se musí spočítat programově (např. součtem řady,

přibližnými vzorci). Aby běžný programátor nemusel tyto výpočty zbytečně programovat sám, dodává výrobce překladače tyto funkce ve formě hotového strojového kódu právě v modulech zvaných knihovny.
A nyní konečně detailně k překladu. Překlad zpravidla probíhá dvoufázově. V první fázi, která se skutečně nazývá *překlad* a provádí ji překladač, se provede syntaktická kontrola. Výsledkem překladu není ještě samostatně spustitelný program, ale zatím jeho „polotovar“. Polotovar se nazývá *object code* a bývá uložen v souboru s příponou .obj, v operačních systémech Unixového typu typicky mají soubory příponu .o. Soubor s obj kódem obsahuje již části kódu s běžnými instrukcemi (např. sčítání). Pokud obsahuje zdrojový kód volání knihovních funkcí (např. sin), je v kódu pouze odkaz na volání funkce sin. Další fáze překladu je *sestavení* (neboli linkování) a provádí jej sestavovací program (linker). Vstupem sestavovacího program je object kód a potřebné knihovny. Sestavovací program vybere z knihoven kódy funkcí použitých v programu, připojí je k object kódu a teprve výsledkem sestavení je hotový spustitelný program.

Úvod do programovacího jazyka C

Historie jazyka

Programovací jazyk C je úzce spjat s operačním systémem UNIX. Jazyk vznikl, stejně jako operační systém, v AT&T Bellových laboratořích v 70. letech a jeho tvůrci jsou pánové Kernighan a Ritchie. Proto se také první verze jazyka jmenuje K&R (1977). Později byly některé rysy jazyka upraveny a doplněny (např. přehlednější způsob deklarace formálních parametrů procedur a funkcí blíže k Pascalu). Výsledkem byl standard ANSI C (1988), který byl dále rozšiřován (poslední je ANSI C 99, je připravován standard ANSI C1X). Standardu ANSI se budeme držet při výuce.

Záměrem autorů při vývoji nového operačního systému UNIX bylo vytvořit jednoduchý a stabilní operační systém. Zároveň s novým operačním systémem chtěli autoři přijít i s novým programovacím jazykem, který by byl úsporný, efektivní (blízký strojovému jazyku) a umožňoval generovat rychlý výsledný kód. Tím jazykem se měl stát a stal se právě jazyk C (název C je údajně pokračováním pracovního označení A a B prvních návrhů jazyka). Autoři uvažovali, že jazyk C se stane výhradním programovacím jazykem pod operačním systémem UNIX (pomocí něhož se budou programovat i některé části operačního systému). První překladač byl implementován právě pod operačním systémem UNIX na minipočítači PDP 11 (obrázek 1) od firmy DEC (Digital Equipment Corporation).

Obrázek 1: Minipočítač PDP 11 (1970)
(na snímku Dennis Ritchie a Kenneth Thompson, tvůrci UNIXu)

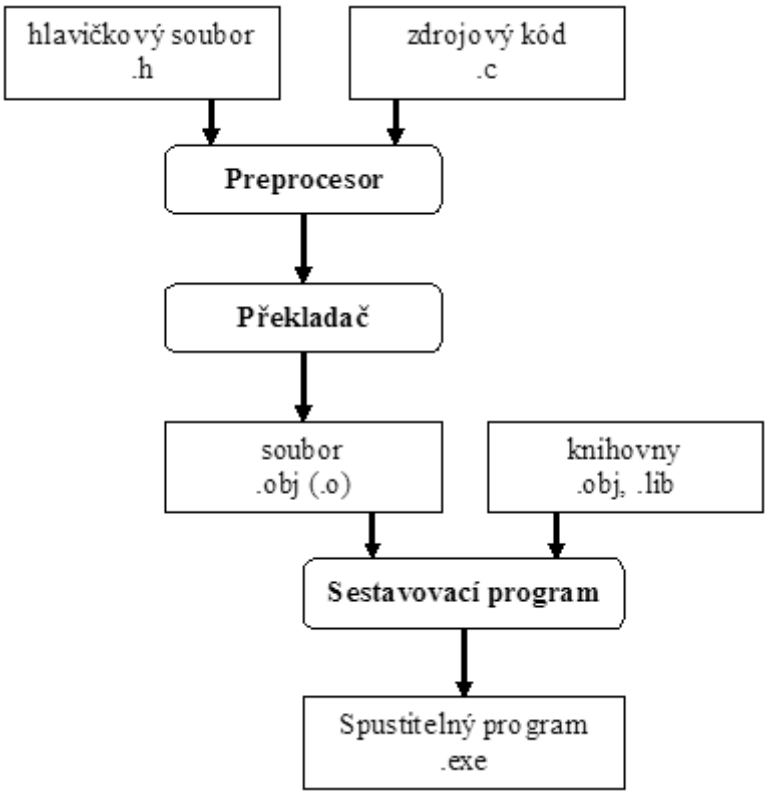
Základní vlastnosti jazyka C

- kompilační jazyk
- univerzální a úsporný programovací jazyk
- slabá typová kontrola
- funkcionální rysy
  - jedna z definovaných funkcí se musí jmenovat `main` a představuje hlavní program.
- **case-sensitive**
  - rozlišují se malá a velká písmena ve zdrojovém kódu.
- každý příkaz je **ukončen** středníkem

Překlad zdrojového kódu v jazyce C

Jazyk C je kompilační jazyk, provádí se překlad ze zdrojového do strojového kódu. Oproti jiným jazykům má překlad určitou zvláštnost. Před vlastní kompilací se provádí ještě tzv. *preprocessing* („předpřeklad“), který spočívá v textových úpravách zdrojového kódu. Úpravy jsou samozřejmě pouze „interní“, v rámci

překladu, soubor se zdrojovým kódem zůstává nedotčen. Preprocessing provádí část překladače, která se přirozeně nazývá *preprocessor* (česky preprocesor). Příkazy pro preprocesor jsou součástí zdrojového kódu a nazývají se *direktivy preprocesoru*. Pro provedení preprocessingu je zdrojový kód přeložen do souboru s „object kódem“ a pak je možné provést sestavení. Princip překladu v jazyce C je na obr. 2.



Obrázek 2: Překlad v jazyce C

Základní direktivy preprocesoru

Direktivy preprocesoru začínaji znakem „#“ a *nejsou ukončeny středníkem*! Nezbytná direktiva, kterou jsme běžně používali již v předmětu Programování, je `#include`. Pomocí této direktivy se vkládá do zdrojového kódu soubor s hlavičkami funkcí (soubor s příponou .h); takto lze samozřejmě vložit jakýkoliv textový soubor. Hlavičkových souborů je několik desítek a bývají uloženy zpravidla v podadresáři `include` adresáře s programovým balíkem překladače. My prozatím uvedeme několik základních hlavičkových souborů, které budeme používat:

Název hlavičkového souboru	Popis
`stdio.h`	Standard input-output (standardní vstup a výstup). Obsahuje funkce pro výstup v textovém módu na obrazovku a vstup z

	klávesnice (obecně na standardní výstup/ze standardního vstupu).
`conio.h`	Console input-output. Doplňkové funkce pro textový vstup/výstup (barvy, posuvy kurzoru, pro PC a DOS).
`stdlib.h`	Standard library. Obsahuje některé speciální funkce, např. možnost volání příkazů DOSu z programu, převodní funkce číselných hodnot na textové vyjádření, generátor náhodných čísel aj.
`math.h`	Matematické funkce a konstanty.

Programovací jazyk C byl navržen ve svém základě jako velmi úsporný, aby jádro jazyka bylo co nejméně závislé na cílové platformě. Proto nejsou součástí jazyka ani příkazy pro vstup a výstup na obrazovku, ale existují ve formě funkcí v externích knihovnách. Základní funkce jsou uvedeny v souboru `stdio.h`, bez něhož bychom nenapsali žádný program v jazyce C, který by vypisoval cokoliv na obrazovku.

Hlavičkový soubor vložíme uvedením direktivy na počátku zdrojového kódu:

```
#include <stdio.h>
```

Pokud jméno hlavičkového souboru vložíme mezi znaky < a >, hledá preprocesor hlavičkový soubor ve standardní adresáři překladače s hlavičkovými soubory (zpravidla také adresář s názvem include). Je-li jméno souboru mezi uvozovkami, např. `#include "seznam.h"`, hledá překladač soubor v aktuálním adresáři projektu. Lze uvést i cestu, syntaxe se řídí daným operačním systémem.

Další velmi využívaná direktiva je `#define`. Umožňuje definovat tzv. *symbolickou konstantu* (jazyk C neměl ve verzi K&R klíčové slovo `const`, to bylo doplněno až do ANSI C), např. `#define MAX 10`. Preprocesor nahradí ve zdrojovém kódu všechny výskyty řetězce MAX řetězcem 10. Náhradu ilustruje následující příklad, včetně direktivy `#include`.

Příklad:

Předpokládejme, že soubor *definice.txt* obsahuje jeden řádek:

```
float obsah(float a, float b);
```

Zdrojový kód má podobu:

```
#include "definice.txt"
#define MAX 10

int main(int argc, char **argv)
```

```
{
  ...
    if (x > MAX) printf("Limit je %d",MAX);
  ...
}
```
Tedy, po zpracování preprocesorem bude mít zdrojový kód podobu:
```
float obsah(float a, float b);

int main(int argc, char **argv)
{
  ...
    if (x > 10) printf("Limit je %d",10);
  ...
}
```
Opět připomínám, že původní zdrojový kód zapsaný programátorem se nemění, zpracování preprocesorem je pouze „vnitřní“, programátor výsledek normálně nevidí. Je však možné u většiny překladačů nechat provést pouze zapracování preprocesorem a výsledek zobrazit, resp. uložit (viz dále).

Direktiva `#define` slouží také k definici makra. Ukážeme definici makra SOUCET s parametry, které počítá součet dvou čísel:

```
#define SOUCET(a,b) (a+b)

int main(int argc, char **argv)
{
  int x;
  ...
    x = 3*SOUCET(2,3);
  ...
}
```
Preprocesor přímo nahradí ve zdrojovém kódu výskyt SOUCET(2,3) řetězcem (2+3):
```
int main(int argc, char **argv)
{
  int x;
  ...
    x = 3*(2+3);
  ...
}
```
Závorky v definici makra `(a+b)` jsou důležité. Pokud bychom je v definici makra nenapsali, tedy zápis by vypadal `#define SOUCET(a,b) a+b`, preprocesor by provedl náhradu přesně podle definice, tj. `x = 3*2+3;`. Vzhledem k prioritě operátorů je výsledný výraz jiný, než programátor zřejmě původně zamýšlel. Při definici makra je tedy potřebné uvědomit si všechny souvislosti.

Pomocí zmíněné direktivy je možné definovat pouze symbol (identifikátor), např. `#define PRACOVNI`. Využijeme jej v podmíněném překladu, kdy pomocí direktiv `#ifdef`, resp. `#ifndef` řídíme, které části kódu má preprocesor vypustit a které ponechat. Ukážeme si program s podmíněným překladem, kdy využijeme tento princip při ladění. Definujeme symbol LADENI a do kódu napíšeme

podmínku v podobě direktivy preprocesoru `#ifdef`. Do ní vložíme kontrolní výpis, který bude ve výsledném programu po odladění vynechán.

```
#include <stdio.h>
#define LADENI

int main(int argc, char **argv)
{
  int x;

  printf("Zadej cele cislo");
  scanf("%d",&x);
  #ifdef LADENI
    printf("Zadal jsi %d\n",x);
  #endif
  ...
  return 0;
}
```

Poznámka: volání funkce printf nesmí být na stejném řádku jako direktiva.

Protože je symbol LADENI předem definován, preprocesor ponechá v textu volání funkce printf. Po ukončení ladění programu direktivu `#define LADENI` zakomentujeme a provedeme ještě jednou překlad. Pracovní výpis pak v programu nebude.

Další využití podmíněného překladu spočívá v možnosti řídit překlad podle různých požadavků. Předpokládejme, že máme nějaké dvě knihovny (dva hlavičkové soubory), jedna obsahuje definice datových typů s optimalizací pro 32 bitový procesor (vypocty32.h) a druhá datové typy s optimalizací pro 64 bitový procesor (vypocty64.h). Chceme napsat univerzální program, který bychom mohli bez složitých úprav jednoduše přeložit buď ve 32 nebo 64 verzi. Pomocí direktiv to provedeme snadno:

```
#define PREKL32

#ifdef PREKL32
  #include "vypocty32.h"
#else
  #ifdef PREKL64
    #include "vypocty64.h"
    #else
      #error Musi byt definovan symbol PREKL32 nebo PREKL64
  #endif
#endif

...
```

Pokud zapomeneme symbol definovat, preprocesor vykoná direktivu `#error` - vypíše chybové hlášení a zastaví překlad.

Programy budeme vyvíjet a ladit ve volně šířitelných prostředích Dev C++ nebo CodeBlocs (obě jsou založena na volně šířitelném překladači `gcc`). Prostředí Dev C++ je možné stáhnout z adresy http://www.bloodshed.net/devcpp.html prostředí

CodeBlocks lze stáhnout z http://www.codeblocks.org. V případě prostředí Codeblocks stahujte instalační soubor, který má v názvu mingw, jinak se Vám nainstataluje pouze prostředí bez překladače. Volně šířitelné prostředí DevC++ má méně komfortní možnosti krokování programů.

Výhody podmíněného překladu si ještě ukážeme v souvislosti s dvěma překladači používanými při výuce. Představme si program, který pracuje s náhodnými čísly. Knihovna `stdlib.h` překladače CodeGear obsahuje definici procedury pro inicializaci generátoru náhodných čísel `randomize()` a makro `random(num)` vracející pseudonáhodné číslo z rozsahu 0 až n-1. Funkce `randomize()` využívá pro inicializaci generátoru funkci `srand()`, která má za parametr funkci `time()` vracející aktuální čas v počtu sekund od 1.1.1970. Funkce `time()` je z knihovny `time.h`. Makro `random(n)` je založeno na funkci `srand()`, která vrací náhodné číslo v rozsahu 0 až RAND_MAX (konstanta závislá na překladači). Překladač Dev C++ funkce `randomize()` a `random()` v knihovně definovány nemá. Programátor musí definovat funkci a makro sám. Pokud chce programátor mít kód přenositelný, tj. přeložitelný v obou překladačích s minimem úprav, využije s výhodou právě podmíněný překlad. Má-li programátor v kódu vlastní definici zmiňované procedury a makra (kód připravený pro Dev C++) a chce jej přeložit pod překladačem CodeGear, hlásí překladač chybu opakované definice (samozřejmě za předpokladu, že také vkládáme knihovnu `stdlib.h`). Jedna z cest je pro CodeGear vymazat definice nebo je zakomentovat. Elegantnější je uzavřít definice do podmíněného překladu a zakomentovat pouze definici symbolu např. DEVCPP:

```
#define DEVCPP

#include <stdio.h>
#include <stdlib.h>
#ifdef DEVCPP
#include <time.h>
#endif

#define MAX 10

#ifdef DEVCPP
#define random(num)  (rand()%num)
void randomize(void) { srand((unsigned) time(NULL)); }
#endif

int main()
{
  ...
}
```

Příklad počítající četnosti náhodných čísel, který byl probírán ve cvičení předmětu

Komentáře

Komentáře (poznámky) se v jazyce C uzavírají mezi dvojice elementů `/* */`. Komentáře mohou být víceřádkové a nesmějí být vnořeny (tzv. nested comments)! Komentáře jsou odstraněny preprocesorem.

Většina současných překladačů, které jsou zároveň překladači jazyka C++, dovolují používat jednořádkové komentáře ve stylu C++, tj. uvozené `//`. Tyto komentáře platí do konce řádku a nemají ukončující element.