# Zadání projektu z předmětů IFJ a IAL

Luboš Lorenc
email: lorenc@fit.vutbr.cz

25. září 2005

## 1 Obecné informace

**Název projektu:** Implementace překladače imperativního jazyka *IFJ05*.

**Konzultace:**

- Ing. Luboš Lorenc (vhodné si dopředu přes e-mail zarezervovat)
  - pondělí od 13:15 do 14:50
- Ing. Rudolf Schönecker (vhodné si dopředu přes e-mail zarezervovat)
  - středa od 9:00 do 11:00

**Datum a způsob odevzdání:** Do 20. prosince 2005 do 13:00, pouze prostřednictvím IS FIT do datového skladu předmětu IFJ.

**Řešitelské týmy:**

- Projekt budou řešit čtyř až pětičlenné týmy.
- Registrace do týmů se provádí zápisem na příslušnou variantu zadání v IS FIT.
- Týmy s jiným počtem členů jsou nepřípustné a nebudou hodnoceny.
- Zadání obsahuje více variant. Každý tým řeší pouze jednu vybranou variantu. Výběr varianty se provádí přihlášením do skupiny řešitelů dané varianty v IS FIT. Převážná část zadání je pro všechny skupiny shodná a jednotlivé varianty se liší pouze ve způsobu implementace tabulky symbolů a vestavěné funkce interpretu. V IS je pro označení variant použit zápis `číslice(písmeno)/číslice`, kde číslice spolu s písmenem před lomítkem udávají variantu vestavěné funkce (viz kapitola 2.1.4) a číslice za lomítkem udává variantu řešení tabulky symbolů (viz kapitola 2.2.2).

**Hodnocení:**

- Do předmětu IFJ získá každý člen řešitelského týmu maximálně 20 bodů za programovou část projektu a 5 bodů za dokumentaci.
- Do předmětu IAL získá každý člen řešitelského týmu maximálně 10 bodů za prezentaci řešení a 5 bodů za dokumentaci.

- Získané body budou mezi jednotlivé členy týmu děleny rovnoměrně. Členové řešitelského týmu však mají právo stanovit poměr dělení bodů za programovou část a dokumentaci jinak. Podrobné informace jsou uvedeny v kapitole 3.2. Případné změny v rozdělení bodů za prezentaci je nutné řešit s hodnotící komisí.
- Dokumentace je k programu, takže za samotnou dokumentaci lze získat maximálně jeden bod.
- Za tvůrčí přístup (různá rozšíření a podobně) lze získat body navíc, maximálně však 25%.

# 2 Zadání

Vytvořte program, který načte zdrojový soubor v jazyce IFJ05, jež představuje 3-adresový kód s výrazy (definovaný níže) a interpretačním způsobem ho vyhodnotí. Pokud v průběhu vyhodnocení zjistí, že ve vstupním souboru je chyba, tak to oznámí vhodným způsobem (určí typ chyby) a ukončí činnost.

Jméno vstupního souboru bude předáno jako první a jediný parametr na příkazové řádce s tím, že je možné zadat jej i s relativní, či absolutní cestou k němu. Program bude směrovat všechny výstupy na standardní výstup, všechna chybová hlášení na standardní chybový výstup a všechny vstupy přejímat ze standardního vstupu, tj. bude se jednat o konzolovou aplikaci, nikoliv o aplikaci s GUR.

Veškeré regulární výrazy použité v dalším textu jsou vždy zapsány podle pravidel GNU pro rozšířené regulární výrazy. Případné mezery v regulárních výrazech jsou většinou (kromě případů, kdy jsou nutné pro oddělení dvou po sobě jdoucích slov) sázeny pouze pro zlepšení čitelnosti a nejsou součástí regulárního výrazu. Klíčová slova jsou uzavřena v apostrofech, přičemž znak apostrof není součástí jazyka! Bližší popis regulárních výrazů lze nalézt například v systému GNU/Linux po zadání příkazu `info grep`.

## 2.1 Popis programovacího jazyka

### 2.1.1 Obecné vlastnosti a datové typy

- Rozlišují se velká a malá písmena.
- Identifikátor je definován regulárním výrazem `[a-zA-Z][a-zA-Z0-9_]*`
- Celočíselný literál (konstanta) je definován výrazem `[0-9]+`
- Desetinný literál (dvojnásobná přesnost) je definován regulárním výrazem `[0-9]+\.[0-9]*(e[+-]?[0-9]+)?`
- Řetězcové literály (řetězce) jsou ohraničené uvozovkami (vložené uvozovky se uvozují zpětným lomítkem `\"`, znak konce řádku označuje dvojice `\n`, zpětné lomítko se zdvojuje — tedy například řetězec `"On řekl: \"Ahoj, co děláš?\""` bude odpovídat textu *On řekl: „Ahoj, co děláš?“*). Řetězec nesmí obsahovat řídicí znaky (znaky s číselným kódem menším než $20_H$).
- Programovací jazyk IFJ05 je netypovaný. To znamená, že jednotlivé proměnné nemají předem určený datový typ a mohou uchovávat hodnotu libovolného typu.

### 2.1.2 Příkazy a návěští

Program je tvořen *neprázdnou* posloupností příkazů a návěští v libovolném pořadí. Každý příkaz je ukončen znakem `';'` (středník). Každé návěští je ukončeno dvojznakem `':::'` (dvě dvojtečky

bez mezery mezi znaky). Pro návěští platí stejná pravidla jako pro identifikátory s tím, že v programu se nesmí vyskytovat návěští a identifikátor stejného znění. Znaky `' '` (mezera), `'\n'` (nový řádek), `'\t'` (tabelátor), `'\r'` (návrat vozíku) a `'\f'` (nová stránka) slouží pro oddělení jednotlivých částí příkazů, či pro formátování vstupního textu a nemají žádný vztah k interpretaci. Přesto se mohou v libovolném množství a kombinacích ve vstupním textu vyskytovat mezi libovolnými částmi vstupního textu.

Vstupní program může obsahovat tyto příkazy:

- (Re)definice proměnné, tvaru: `<identifikátor_proměnné> := <výraz>`
  - proměnná vpravo v příkazu (v rámci výrazu) musí již mít známou hodnotu, musí být definována,
  - proměnná vlevo v příkazu je nově definována, nebo redefinována.
- Skok, tvaru: `'go_to' <návěští>` (návěští zde není následováno dvojznakem "`::`").
- Podmíněný skok, tvaru: `'on' <výraz> 'go_to' <návěští>`
  - vstupem může být libovolný neprázdný výraz,
  - skok se provede tehdy, pokud číselný výraz bude nenulový, nebo v případě řetězce bude tento neprázdný.
- Tisk na standardní výstup, tvaru: `'print' <identifikátor_proměnné>`
  - vstupem může být libovolná proměnná, ale musí mít předem známou hodnotu.
- Čtení ze standardního vstupu, tvaru: `'read' <identifikátor_proměnné>`
  - na vstupu nepředpokládejte chybu v zadání — případnou detekci a vhodné ošetření takových chyb je však možné brát jako rozšíření,
  - parametrem může být libovolná proměnná, ani nemusí mít předem známou hodnotu, takže tímto dojde k její definici.

### 2.1.3 Výrazy

Výrazy jsou tvořeny konstantami, již definovanými proměnnými, závorkami, voláním vestavěné funkce (viz kapitola 2.1.4) a binárními operátory: $+$, $-$, $*$, $/$, $<$, $>$, $=$, $!=$, $<=$, $>=$. Platí tato pravidla:

- Pokud se jedná o jeden z těchto operátorů: $+$, $-$, $*$, $/$, potom platí, že význam těchto operátorů pro číselné operandy je shodný s pravidly jazyka ANSI-C. Pro řetězcové operandy je definován pouze operátor $+$ ve významu konkatenace řetězců, přičemž jejich délka není omezena (jen velikostí virtuální paměti počítače).
- Jedná-li se o jeden z těchto operátorů: $<$, $>$, $=$, $!=$, $<=$, $>=$, potom význam těchto operátorů je standardní s tím, že pro řetězce je definována pouze rovnost a nerovnost. Výsledkem porovnání je celočíselná hodnota, která je rovna 1, pokud je podmínka splněna a 0 v opačném případě.
- Operátory mají tuto posloupnost vzájemných priorit (nejvyšší na začátku):

  závorky, volání funkce
  $*$, $/$
  $+$, $-$
  $<$, $>$, $<=$, $>=$
  $=$, $!=$

### 2.1.4 Vestavěné funkce

Vestavěné funkce jsou součástí variantního zadání projektu. Jedná se o implementaci různých algoritmů řazení. Každý tým implementuje pouze funkci odpovídající jeho zadání. Případnou implementaci ostatních funkcí lze provést jako rozšíření. Funkce, kterou máte implementovat, je určena číslicí a písmenem před lomítkem v označení varianty zadání. Číslice určuje, zda se jedná o řazení čísel nebo řetězců a písmeno pak označuje použitý algoritmus. Tedy například označení „2(b)“ znamená, že budete implementovat algoritmus Radix sort pro řazení řetězců. Všechny funkce jsou volány svým identifikátorem následovaným seznamem parametrů v kulatých závorkách `id \( (výraz (, výraz)*)? \)` . Každý parametr je tvořen libovolným výrazem (může opět obsahovat volání funkce) a jednotlivé parametry jsou vzájemně odděleny čárkami. Počet parametrů je ve všech případech neomezený, ale vždy musí být minimálně jeden.

1. **Řazení čísel**
   Každý parametr může být buď celočíselný nebo desetinný. Výsledkem je řetězec obsahující seřazenou číselnou posloupnost. Jednotlivá čísla v tomto seznamu následují bezprostředně po sobě a jsou vždy oddělena čárkou. Za posledním číslem čárka není. V seřazeném řetězci se nevyskytují žádné redundantní znaky (mezery, konce řádků, a podobné, které nejsou součástí původních řetězců).

   (a) **Quicksort** — Pro řazení použijte algoritmus Quick sort.
   (b) **Radixsort** — Pro řazení použijte algoritmus Radix sort.
   (c) **Heapsort** — Pro řazení použijte algoritmus Heap sort.
   (d) **Shellsort** — Pro řazení použijte algoritmus Shell sort.
   (e) **Mergesort** — Pro řazení použijte algoritmus Merge sort.

2. **Řazení řetězců**
   Každý parametr je typu řetězec. Výsledkem je řetězec obsahující abecedně seřazenou posloupnost vstupních řetězců. Řazení bude provedeno podle pravidel pro české abecední řazení. V seřazeném řetězci se nevyskytují žádné redundantní znaky (mezery, konce řádků, a podobné, které nejsou součástí původních řetězců).

   (a) **Quicksort** — Pro řazení použijte algoritmus Quick sort.
   (b) **Radixsort** — Pro řazení použijte algoritmus Radix sort.
   (c) **Heapsort** — Pro řazení použijte algoritmus Heap sort.
   (d) **Shellsort** — Pro řazení použijte algoritmus Shell sort.
   (e) **Mergesort** — Pro řazení použijte algoritmus Merge sort.

## 2.2 Implementace interpretu

Při implementaci využijte znalosti konečných automatů, precedenční syntaktické analýzy a programovacích technik. Naopak je zakázáno využít konstruktorů lex/flex, yacc/bison a podobných. Stejně tak není možné využít techniky rekurzivního sestupu, nebo rozkladové tabulky a zásobníkového automatu, krom využití pro precedenční syntaktickou analýzu a zpracování parametrů funkcí (například pokud je parametrem jedné funkce jiná funkce). V případě pochybností, zda postupujete správně, se včas informujte, neboť na pozdější „nářky“ nebude brán zřetel. Kromě toho není povoleno využít prvků objektově orientovaného návrhu a implementace. Toto je podmíněno i tím, že implementace bude provedena v jazyce ANSI-C. Návrh implementace interpretu je zcela v režii řešitelských týmů s tím, že je nutné dodržet následující specifikace týkající se typových kontrol a tabulky symbolů.

### 2.2.1 Typová kontrola

Jazyk IFJ05 je netypovaný. Interpret však bude provádět run-time typování všech proměnných. Pro typování bude využívat následujících datových typů:

- Celočíselný (32 bitů),
- desetinný (minimálně dvojnásobná přesnost),
- řetězec (bez omezení délky).

Typ proměnné se vždy určuje podle typu výrazu, který je do ní při definici přiřazen. Výsledný typ výrazu se postupně určuje v průběhu jeho vyčíslování podle následujících pravidel:

- Typ literálu (konstanty) určuje lexikální analyzátor podle jeho tvaru,
- typ proměnné je znám od okamžiku její definice,
- typ literálu načítaného ze standardního vstupu pomocí `'read'` se určuje podle stejných pravidel jako typ literálu v textu programu,
- typ výsledku operací porovnání je vždy celočíselný,
- typ výsledku ostatních operací se řídí následujícími pravidly:
  - Jsou-li oba operandy stejného typu, je výsledek stejného typu jako jsou oba operandy,
  - je-li typ jednoho operandu celočíselný a typ druhého desetinný, je typ výsledku desetinný,
  - je-li typ jednoho operandu celočíselný nebo desetinný a typ druhého řetězec, potom:
    - Pokud se jedná o operaci sčítání, tak se neřetězcový operand převede na řetězec a výsledkem je konkatenace obou řetězců (první je ten, který byl ve výrazu více vlevo). Do výsledného řetězce nejsou přidány žádné redundantní znaky (například mezery).
    - Pokud se jedná o jakoukoliv jinou operaci, dochází k chybě. Tato chyba bude oznámena a interpretace se ukončí.

### 2.2.2 Tabulka symbolů

Tabulku symbolů implementujte metodou, odpovídající označení Vašeho zadání:

1. Pomocí binárního vyhledávacího stromu,
2. pomocí AVL stromu,
3. pomocí hashovací tabulky.

# 3 Instrukce ke způsobu vypracování a odevzdání

## 3.1 Obecné informace

Za celý tým odevzdává projekt jediný student. Všechny odevzdávané soubory budou programem TAR+GZIP či ZIP zkomprimovány do jediného archivu, který se bude jmenovat přihlašovací_jméno.tgz, či přihlašovací_jméno.zip. Archiv nesmí obsahovat adresářovou strukturu a speciální či spustitelné soubory. Všechny názvy souborů uvnitř archivu budou náležet do jazyka

definovaného tímto regulárním výrazem `[a-z_.0-9]+`. Řešení budou zpracována automatizovanými prostředky a při nedodržení těchto pokynů nebude řešení zpracováno a bude považováno za neodevzdané. Archivní soubor bude odevzdán přes IS FIT.

Dále je třeba odevzdat celý projekt v daném termínu (viz výše). Pokud tomu tak nebude, je projekt opět považován za neodevzdaný, nebo minimálně se dá očekávat ztráta bodů dle toho, o jaké zpoždění půjde. Stejně tak, pokud se bude jednat o plagiátorství jakéhokoliv druhu, je projekt považován za neodevzdaný.

## 3.2 Dělení bodů

Získané body budou standardně rovnoměrně děleny mezi všechny členy týmu. Pokud se členové týmu dohodnou na jiném dělení, přiloží do archivu se svým řešením soubor `rozdeleni` s následujícím pevným formátem. Na každém řádku je uveden login jednoho člena týmu, bez mezery je následován dvojtečkou a po ní je bez mezery uveden požadovaný počet procent bodů bez uvedení znaku `%`. Každý řádek (i poslední) je poté ihned ukončen jedním znakem `<LF>`. Jeden řádek tohoto souboru tedy bude vypadat zhruba takto: `xnovak99:25`. Součet všech uvedených procent musí být roven 100. *V případě nedodržení formátu tohoto souboru nebo jeho nezačlenění do odevzdaného řešení bude použito rovnoměrné dělení bodů bez možnosti reklamace rozdělení.* Zároveň platí, že žádný z členů týmu nemůže získat více bodů, než je maximální bodové ohodnocení projektu.

## 3.3 Požadavky na řešení

### 3.3.1 Textová část řešení

Součástí řešení bude dokumentace, vypracovaná ve formátu PDF či PS — jiný formát není povolen. Dokumentace bude vypracována v **českém nebo anglickém jazyce** v rozsahu cca 4–5 stran A4. Tato dokumentace bude obsahovat:

- Jména, příjmení a přihlašovací jména řešitelů,
- popis vašeho způsobu řešení kompilátoru (především z pohledu předmětu IFJ),
- popis vašeho způsobu řešení řadicího algoritmu a tabulky symbolů (především z pohledu předmětu IAL),
- upozornění na případná rozšíření proti zadání — dle implementace.

Dokumentace nesmí obsahovat:

- Kopii zadání,
- text, který není váš původní (kopie z přednášek, sítě, WWW, ...).

Dokumentace bude uložena v jednom souboru `dokumentace.xxx`, kde xxx je buďto `pdf` nebo `ps`. Jakékoliv jiné formáty dokumentace, než předepsané, budou ignorovány, což povede ke ztrátě bodů za dokumentaci.

### 3.3.2 Programová část řešení

Programová část řešení bude vypracována v jazyce (ANSI) C **bez** použití generátorů lex/yacc/bison či jiných podobného ražení. Programy musí být přeložitelné překladačem gcc. Při hodnocení budou projekty překládány pod OS GNU/Linux. Počítejte tedy s touto skutečností (především, pokud budete projekt psát pod jiným OS). Pokud projekt nepůjde přeložit či nebude správně pracovat kvůli použití funkce nebo nějaké nestandardní implementační techniky závislé na OS, ztrácíte právo na reklamaci výsledků. Ve sporných případech bude vždy za

platný považován výsledek překladu na serveru merlin **bez použití jakýchkoliv dodatečných nastavení** (proměnné prostředí, ...).

Součástí řešení bude soubor `Makefile` (projekty budou překládány pomocí `gmake` [1]).

Binární soubor (přeložený interpret) v žádném případě neposílejte v archivu (jednak bude archiv zbytečně velký a binární soubor bude stejně okamžitě smazán, navíc můžete být postihnuti ztrátou bodů). Pokud soubor pro sestavení cílového programu nebude obsažen, nebo se na jeho základě nepodaří sestavit cílový program, bude se projekt hodnotit jako neodevzdaný.

Hlavičky zdrojových textů musí obsahovat název projektu, přihlašovací jména a jména všech řešitelů. Pokud tomu tak nebude, bude projekt hodnocen automaticky 0 body.

Veškerá chybová hlášení vzniklá v průběhu činnosti interpretu budou vždy vypisována na standardní chybový výstup. Veškeré texty požadované řídicím programem budou vypisovány na standardní výstup. Kromě chybových hlášení **nebude** interpret v průběhu interpretace na žádný výstup vypisovat **žádné** znaky či dokonce celé texty, které nejsou přímo předepsány řídicím programem. Testování bude probíhat pomocí automatu, který bude postupně spouštět sadu testovacích programů v odevzdaném interpretu a porovnávat produkované výstupy s výstupy očekávanými. Pro porovnání výstupů bude použit program `diff` (viz `info diff`). Znovu připomínám, že jediný neočekávaný znak, který váš interpret svévolně vytiskne, povede k nevyhovujícímu hodnocení aktuálního testu a tím snížení bodového hodnocení celého projektu.

*Nedodržení těchto podmínek se bude hodnotit jako neodevzdaný projekt.*

### 3.4 Konzultace

Konzultace budou probíhat ve výše uvedených termínech a ve výjimečných případech i jindy. Doporučuji se na ně předem domluvit. Domluva předem pomůže řešit případný zájem více jedinců o konzultaci ve stejný čas a taktéž se takto ujistíte o mé připravenosti.

*V případě jakýchkoliv nejasností ohledně řešení neváhejte a kontaktujte mě. Drobné dotazy lze většinou rychle řešit e-mailem.*

### 3.5 Zásady bodového hodnocení

Projekt je hodnocen jako neodevzdaný, t.j. 0 body, pokud:

- Nebude odevzdán včas (za každý započatý den zpoždění minimálně 20% bodů dolů),
- nebude odevzdán v předepsaném formátu,
- nebude možné sestavit program tak, jak je uvedeno v zadání (např. je nutný další zásah do předpisů pro sestavení, ruční nastavení přístupových cest, ...),
- půjde o plagiát v jakékoliv podobě,
- výstup programu nebude odpovídat zadání,
- ...

### 3.6 Jak postupovat při řešení projektu

Při řešení je pochopitelně možné využít vlastní výpočetní techniku. Instalace překladače gcc není nezbytně nutná, pokud máte jiný překladač jazyka C již instalován a nehodláte využívat vlastnosti, které právě tyto a žádné jiné překladače nemají. Před použitím nějaké *vyspělé* konstrukce je dobré si ověřit, že jí disponuje i překladač, který nakonec bude použit při opravování.

[1] Makefile je soubor, který zpracovává program gmake. V žádném případě to není shellový skript. Více informací lze najít například v `info make` nebo na `http://www.gnu.org`

Po vypracování je též vhodné vše ověřit na cílovém překladači, aby při kontrole vše proběhlo bez problémů.

Pro vlastní řešení je dobré postupovat tak, aby váš produkt co nejdříve generoval nějaké výstupy pro nějaké jednoduché výrazy. Tj. něco, co například dokáže přeložit součet dvou čísel a tisknout výsledek. To totiž umožní hodnotit projekt jako celek a přiznat více, než jen symbolické body.

Takový přístup je možné například realizovat tak, že nejdříve uděláte lexikální analyzátor, poté uděláte syntaktický analyzátor pro zpracování příkazů, a velmi jednoduchý syntaktický analyzátor pro výrazy, který zatím zvládá pouze načítání konstant a práci s jedinou proměnnou. I tak už ale máte interpret, který umí alespoň načítat data ze vstupu a vypisovat na výstup, což už na něco stačí.

Poté vytvoříte tabulku symbolů a přidáváte další syntaktické a sémantické konstrukce do výrazů a tím rozšiřujete funkčnost interpretu.

## 3.7 Doplňující podmínky

Získání minimálně 5 bodů z programové části projektu je nutnou (nikoliv však postačující) podmínkou pro získání zápočtu a tím i pro přístup k závěrečné zkoušce z předmětu IFJ. Garant předmětu IFJ si rezervuje právo na vysvětlení a ústní přezkoušení studentů, kteří dosáhnou zanedbatelného bodového zisku z projektu. Toto přezkoušení bude sloužit k ověření alespoň elementárních praktických schopností získaných v předmětu IFJ.

Při opravování budou projekty překládány překladačem gcc. V současné době se jedná o verzi gcc 3.4.4, ale v době opravování to pravděpodobně bude již verze 4.0.x.

## 3.8 Stanovisko k akademické nečestnosti

*Pokud se při vytváření projektu dopustíte jakéhokoliv přestupku proti akademické morálce (např. plagiátorství), váš přestupek bude ohlášen garantovi předmětu IFJ, který je současně předseda Disciplinární komise FIT VUT. Takovýto podvod typicky vyústí v neudělení zápočtu a tím k neúspěšnému absolvování předmětu IFJ či IAL; výslovně však upozorňujeme, že může vést k vyloučení z akademické obce a tím pádem z celého VUT.*

# Obsah