

ZADÁNÍ PROJEKTU Z PŘEDMĚTŮ IFJ A IAL

Zbyněk Křivka, Radim Kocman, Dominika Klobučníková

email: {krivka, kocman, iklobucnikova}@fit.vut.cz

28. září 2022

1 Obecné informace

Název projektu: Implementace překladače imperativního jazyka IFJ22.
Informace: diskuzní fórum a Moodle předmětu IFJ.
Pokusné odevzdání: čtvrtek 24. listopadu 2022, 23:59 (nepovinné).
Datum odevzdání: středa 7. prosince 2022, 23:59.
Způsob odevzdání: prostřednictvím StudIS, aktivita „Projekt - Registrace a Odevzdání“.

Hodnocení:

- Do předmětu IFJ získá každý maximálně 25 bodů (15 celková funkčnost projektu (tzv. programová část), 5 dokumentace, 5 obhajoba).
- Do předmětu IAL získá každý maximálně 15 bodů (5 celková funkčnost projektu, 5 obhajoba, 5 dokumentace).
- Max. 35 % bodů Vašeho individuálního hodnocení základní funkčnosti do předmětu IFJ navíc za tvůrčí přístup (různá rozšíření apod.).
- **Udělení zápočtu z IFJ i IAL je podmíněno získáním min. 20 bodů v průběhu semestru. Navíc v IFJ z těchto 20 bodů musíte získat nejméně 4 body za programovou část projektu.**
- Dokumentace bude hodnocena nejvýše polovinou bodů z hodnocení funkčnosti projektu, bude také reflektovat procentuální rozdělení bodů a bude zaokrouhlena na celé body.
- Body zapisované za programovou část včetně rozšíření budou také zaokrouhleny a v případě přesáhnutí 15 bodů zapsány do termínu „Projekt - Bonusové hodnocení“ v IFJ.

Řešitelské týmy:

- Projekt budou řešit čtyřčlenné týmy. Týmy s jiným počtem členů jsou nepřípustné (jen výjimečně tříčlenné).
- Vytváření týmů se provádí funkcionalitou Týmy v IS VUT. Tým vytváří vedoucí a název týmu bude automaticky generován na základě loginu vedoucího. Vedoucí má kontrolu nad složením týmu, smí předat vedení týmu jinému členovi a bude odevzdávat výsledný archiv. Rovněž vzájemná komunikace mezi vyučujícími a týmy bude probíhat nejlépe prostřednictvím vedoucích (ideálně v kopii dalším členům týmu).

- Pokud bude mít tým dostatek členů, bude mu umožněno si zaregistrovat jednu ze dvou variant zadání (viz sekce 7) v aktivitě „Projekt - Registrace a Odevzdání“.
- Všechny hodnocené aktivity k projektu najdete ve StudIS, studijní informace v Module předmětu IFJ a další informace na stránkách předmětu¹.

2 Zadání

Vytvořte program v jazyce C, který načte zdrojový kód zapsaný ve zdrojovém jazyce IFJ22 a přeloží jej do cílového jazyka IFJcode22 (mezikód). Jestliže proběhne překlad bez chyb, vrací se návratová hodnota 0 (nula). Jestliže došlo k nějaké chybě, vrací se návratová hodnota následovně:

- 1 - chyba v programu v rámci lexikální analýzy (chybná struktura aktuálního lexému).
- 2 - chyba v programu v rámci syntaktické analýzy (chybná syntaxe programu, chybějící hlavička, atp.).
- 3 - sémantická chyba v programu – nedefinovaná funkce, pokus o redefinice funkce.
- 4 - sémantická/běhová chyba v programu – špatný počet/typ parametrů u volání funkce či typ návratové hodnoty z funkce.
- 5 - sémantická chyba v programu – použití nedefinované proměnné.
- 6 - sémantická/běhová chyba v programu – chybějící/přebývajících výraz v příkazu návratu z funkce.
- 7 - sémantická/běhová chyba typové kompatibility v aritmetických, řetězcových a relačních výrazech.
- 8 - ostatní sémantické chyby.
- 99 - interní chyba překladače tj. neovlivněná vstupním programem (např. chyba alokace paměti atd.).

Překladač bude načítat řídicí program v jazyce IFJ22 ze standardního vstupu a generovat výsledný mezikód v jazyce IFJcode22 (viz kapitola 10) na standardní výstup. Všechna chybová hlášení, varování a ladicí výpisy provádějte na standardní chybový výstup; tj. bude se jednat o konzolovou aplikaci (tzv. filtr) bez grafického uživatelského rozhraní. Pro interpretaci výsledného programu v cílovém jazyce IFJcode22 bude na stránkách předmětu k dispozici interpret.

Klíčová slova jsou sázena tučně a některé lexémy jsou pro zvýšení čitelnosti v apostrofech, přičemž znak apostrofu není v takovém případě součástí jazyka!

3 Popis programovacího jazyka

Jazyk IFJ22 je zjednodušenou podmnožinou jazyka PHP², což je velmi populární skriptovací jazyk používaný při vývoji pro web. Jazyk IFJ22 je navíc doplněn přepínačem PHP, který omezuje implicitní typové konverze u explicitně zadaných typů a aktivuje typové kontroly parametrů.

¹<http://www.fit.vutbr.cz/study/courses/IFJ/public/project>

²Online dokumentace: <http://www.php.net/manual/en/index.php>; na serveru Merlin pod příkazem php8.1 je pro studenty k dispozici verze 8.1.9 z 22. srpna 2022.

3.1 Obecné vlastnosti a datové typy

V programovacím jazyce IFJ22 **záleží** na velikosti písmen u identifikátorů i klíčových slov (tzv. *case-sensitive*)³. A na rozdíl od PHP bez dodatečného přepínače

```
declare(strict_types=1);
```

je IFJ22 jazykem s povinnou typovou anotací formálních parametrů funkcí. Proměnné jsou však stále dynamicky typované, takže typ proměnné určuje její aktuální hodnota.

- *Identifikátor* je definován jako neprázdná posloupnost číslic, písmen (malých i velkých) a znaku podtržítka ('_') začínající písmenem nebo podtržítkem. Jazyk IFJ22 obsahuje navíc níže uvedená *klíčová slova*, která mají specifický význam, a proto se nesmějí vyskytovat jako identifikátory funkcí:

Klíčová slova: **else, float, function, if, int, null, return, string, void, while**

- *Identifikátor proměnné* se skládá z předpony, což je znak \$ (dolar), a identifikátoru⁴.
- *Identifikátor typu* se skládá z volitelné předpony, což je znak ? (otazník), a klíčového slova pro typ (**float, int, string**). Např. **int** nebo **?string**.
- *Celočíselný literál* (rozsah C-int) je tvořen neprázdnou posloupností číslic a vyjadřuje hodnotu celého nezáporného čísla v desítkové soustavě.
- *Desetinný literál* (rozsah C-double) také vyjadřuje nezáporná čísla v desítkové soustavě, přičemž literál je tvořen celou a desetinnou částí, nebo celou částí a exponentem, nebo celou a desetinnou částí a exponentem. Celá i desetinná část je tvořena neprázdnou posloupností číslic. Exponent je celočíselný, začíná znakem 'e' nebo 'E', následuje nepovinné znaménko '+' (plus) nebo '-' (mínus) a poslední částí je neprázdná posloupnost číslic. Mezi jednotlivými částmi nesmí být jiný znak, celou a desetinnou část odděluje znak '.' (tečka)⁵.
- *Řetězcový literál* je oboustranně ohraničen dvojími uvozovkami (" , ASCII hodnota 34). Tvoří jej libovolný počet znaků zapsaných případně i na více řádcích. Možný je i prázdný řetězec (""). Znaky s ASCII hodnotou větší než 31 (mimo ") lze zapisovat přímo. Některé další znaky lze zapisovat pomocí escape sekvence: '\\"', '\\n', '\\t', '\\\\'. Jejich význam se shoduje s odpovídajícími znakovými konstantami jazyka PHP⁶. Na rozdíl od jazyka C nelze escape sekvencí vytvořit chybu – pakliže znaky za zpětným lomítkem neodpovídají žádnému z uvedených vzorů, jsou (včetně lomítka) bez jakýchkoli náhrad součástí řetězce. Expanzi (interpolaci) proměnných v řetězcích neuvažujte, nicméně znak dolar lze sázet jen pomocí escape sekvence '\\\$'. Znak v řetězci může být zadán také pomocí escape sekvence '\\xdd', kde dd je dvoumístné hexadecimální číslo od 01 do FF (písmena A-F mohou být malá i velká). Znak v řetězci může být zadán také pomocí obecné escape sekvence '\\ddd', kde ddd je právě třímístné desítkové číslo od 001 do 255.

Délka řetězce není omezena (resp. jen dostupnou velikostí haldy). Například řetězcový literál

³IFJ22 se v tomto ohledu záměrně liší od PHP, ve kterém jsou názvy funkcí a klíčová slova case-insensitive.

⁴Identifikátor proměnné může po \$ obsahovat i klíčové slovo. Např. **\$return**.

⁵Přebytečné počáteční číslice 0 v celočíselné části či exponentu jsou ignorovány.

⁶<https://www.php.net/manual/en/language.types.string.php>

"Ahoj\n\"Sve'te \\034"

reprezentuje řetězec

Ahoj

"Sve'te \". Neuvažujte řetězce, které obsahují vícebajtové znaky kódování Unicode (např. UTF-8).

- *Datové typy* pro jednotlivé uvedené literály jsou označeny **int**, **float** a **string**. Speciálním případem je hodnota **null**. Typy se nedeklarují u proměnných, ale pouze v signaturách funkcí. Hodnotu **null** mohou nabývat všechny parametry a návratové hodnoty typované s předponou **?**.
- *Term* je libovolný literál (celočíselný, desetinný, řetězcový či **null**) nebo identifikátor proměnné.
- Jazyk IFJ22 podporuje *řádkové* a *blokové komentáře* stejně jako v jazyce PHP. Řádkový komentář začíná dvojicí lomítek (**//**, ASCII hodnota 47) a za komentář je považováno vše, co následuje až do konce řádku.
- Blokový komentář začíná posloupností symbolů **/*** a je ukončen dvojicí symbolů ***/**. Vnořené blokové komentáře neuvažujte.

4 Struktura jazyka

IFJ22 je strukturovaný programovací jazyk podporující definice proměnných a uživatelských funkcí včetně jejich rekurzivního volání. Vstupním bodem prováděného programu je neoznačená nesouvislá sekvence příkazů mezi definicemi uživatelských funkcí, tzv. *hlavní tělo* programu.

4.1 Základní struktura jazyka

Program se skládá z prologu následovaného definicemi funkcí prolínajících se s hlavním tělem programu. V tělech uživatelských funkcí lze potom definovat lokální proměnné, používat příkazy přiřazení, větvení, iterace a volání funkcí.

Prolog⁷ se skládá ze speciální otevírací značky a příkazu aktivujícího přepínač:

```
<?php  
declare(strict_types=1);
```

Před (až na otevírací značku), za (až na uzavírací značku), i mezi jednotlivými tokeny se může vyskytovat libovolný počet bílých znaků (mezera, tabulátor, komentář a odřádkování), takže jednotlivé konstrukce jazyka IFJ22 lze zapisovat na jednom či více řádcích nebo může být na jednom řádku více příkazů jazyka IFJ22. Na úplném konci programu se smí (volitelně) vyskytnout uzavírací značka **?>**.

Struktura definice uživatelských funkcí a také popis jednotlivých příkazů je v následujících sekcích.

⁷Prolog uvádíme kvůli kompatibilitě s programy jazyka PHP 8. Za otevírací značkou se v prologu mohou vyskytovat komentáře a bílé znaky.

4.1.1 Proměnné

Proměnné jazyka IFJ22 jsou pouze lokální (a to i v rámci hlavního těla) a mají rozsah platnosti od místa jejich definice až po konec funkce, ve které byly definovány.

Definice proměnné se provádí pomocí příkazu přiřazení, a tím je spjata i s inicializací proměnné. Detailní syntaxe bude popsána v sekci 4.4. Každá proměnná musí být definována před jejím použitím, jinak se jedná o chybu 5.

4.2 Deklarace a definice uživatelských funkcí

Definice funkce se skládá z hlavičky a těla funkce. Definice funkce je zároveň její deklarací. Každá funkce je definována právě jednou⁸, jinak dochází k chybě 3.

Definice funkce nemusí vždy lexikálně předcházet kódu pro volání této funkce. Uvažujte například vzájemné rekurzivní volání funkcí (tj. funkce `foo` volá funkci `bar`, která opět může volat funkci `foo`). Každá funkce musí být definována před jejím použitím, tzv. *voláním funkce* (příkaz volání je definován níže). Definice funkce však nemusí být lexikálně umístěna před jejím příkazem volání, pokud je volána z jiné funkce, která má příkaz volání umístěn až za definicemi obou funkcí. Jsou tedy umožněna vzájemné rekurzivní volání dvou či více funkcí i bez deklarací funkcí.

Příklad (vzájemná rekurze bez ukončující podmínky):

```
function bar(string $param) : string {
    return foo($param);
}
function foo(string $param):string {
    return bar($param);
}
bar("ahoj");
```

Definice funkce je konstrukce (hlavička a tělo) ve tvaru:

```
function id ( seznam_parametrů ) : návratový_typ {
    sekvence_příkazů
}
```

- Hlavička definice funkce sahá od klíčového slova **function** až po určení návratového typu funkce, pak následuje tělo funkce tvořené sekvencí příkazů (viz sekce 4.4) ve složených závorkách.
- Seznam parametrů je tvořen posloupností definic parametrů oddělených čárkou, přičemž za posledním parametrem se čárka neuvádí. Seznam může být i prázdný. Každá definice parametru obsahuje identifikátor typu parametru a identifikátor parametru:

typ \$identifikátor_parametru

Parametry jsou vždy předávány hodnotou.

Příklad:

```
<?php
declare(strict_types=1);
function concat(string $x, string $y): string {
    $x = $x . $y;
```

⁸Tzv. přetěžování funkcí (angl. *overloading*) není v IFJ22 podporováno.

```

    return $x . " " . $y;
}
$a = "ahoj ";
$ret = concat($a, "svete");
write($ret, $a);
?>

```

- V těle funkce jsou její parametry chápány jako předdefinované lokální proměnné s implicitní hodnotou danou skutečnými parametry, avšak již není zakázána modifikace jejich hodnoty a potažmo typu. Výsledek funkce je dán provedeným příkazem návratu z funkce (viz sekce 4.4).

Každá funkce s návratovou hodnotou vrací hodnotu výrazu z příkazu **return**, avšak v případě chybějící návratové hodnoty kvůli neprovedení žádného příkazu **return**, nebo provedením příkazu **return**, ale s výrazem špatného typu (typ návratové hodnoty neodpovídá návratovému typu funkce), dochází k chybě 4.

Funkce bez návratové hodnoty (návratový typ je **void**, tzv. *void-funkce*) je ukončena po posledním příkazu těla funkce nebo provedením příkazu **return** bez výrazu pro návratovou hodnotu. Definice vnořených funkcí je zakázána.

4.3 Hlavní tělo programu

Zdrojový kód vždy začíná prologem (viz sekce 4.1). Před touto značkou se nesmí nacházet žádné znaky (ani komentáře a bílé znaky). Hlavní tělo programu je neoznačená sekvence příkazů jazyka IFJ22, která se prolíná s definicemi funkcí (na nejvyšší úrovni příkazů, definice funkce tudíž nemůže narušit integritu příkazu, a to ani složeného⁹). Hlavní tělo programu může být tvořeno i prázdnou sekvencí příkazů, kdy je pouze vrácena návratová hodnota programu (možné hodnoty viz kapitola 2). Celá sekvence je ukončena koncem zdrojového souboru nebo volitelnou uzavírací značkou '**?>**', ale za uzavírací značkou nesmí následovat žádný další znak (ani bílý). Struktura jednotlivých příkazů je popsána v následující sekci.

4.4 Syntaxe a sémantika příkazů

Příkazy nekončící složeným příkazem jsou ukončeny středníkem (;).
Dílním příkazem se rozumí:

- *Příkaz přiřazení:*

\$id = výraz ;

Sémantika příkazu je následující: Příkaz provádí přiřazení hodnoty pravého operandu *výraz* (viz kapitola 5) do levého operandu *\$id*. Levý operand musí být vždy pouze proměnná (tzv. l-hodnota).

Sémantika příkazu je následující: Příkaz provádí vyhodnocení výrazu *výraz* (viz kapitola 5) a případné přiřazení jeho hodnoty do levého operandu *\$id*. Levý operand musí být proměnná (tzv. l-hodnota) a po přiřazení bude *\$id* stejného typu jako typ hodnoty výrazu *výraz*. Část '*\$id =*' lze vynechat, takže hodnota vyhodnoceného výrazu *výraz* není nikam přiřazena.

⁹Složený příkaz je sekvence příkazů (i prázdná) uzavřená ve složených závorkách.

- *Podmíněný příkaz:*

```

if ( výraz )
{
    sekvence_příkazů1
}
else
{
    sekvence_příkazů2
}

```

Sémantika příkazu je následující: Nejprve se vyhodnotí daný výraz. Pokud je vyhodnocený výraz pravdivý, vykoná se *sekvence_příkazů*₁, jinak se vykoná *sekvence_příkazů*₂. Pokud výsledná hodnota výrazu není pravdivostní (tj. pravda či nepravda - v základním zadání pouze jako výsledek aplikace relačních operátorů dle sekce 5.1), tak se hodnota **null**, prázdný řetězec, 0 a "0" považují za nepravdu a ostatní hodnoty jako pravda.

- *Příkaz cyklu:*

```

while ( výraz )
{
    sekvence_příkazů
}

```

Příkaz cyklu se skládá z hlavičky a těla tvořeného *sekvencí_příkazů*.

Sémantika příkazu cyklu je následující: Opakuje provádění sekvence dílčích příkazů (může být i prázdná) tak dlouho, dokud je hodnota výrazu pravdivá. Pravidla pro určení pravdivosti výrazu jsou stejná jako u výrazu v podmíněném příkazu.

- *Volání vestavěné či uživatelem definované funkce:*

\$id = název_funkce (seznam_vstupních_parametrů) ;

Seznam_vstupních_parametrů je seznam termů (viz sekce 3.1) oddělených čárkami¹⁰. Seznam může být i prázdný. Sémantika vestavěných funkcí bude popsána v kapitole 6. Sémantika volání uživatelem definovaných funkcí je následující: Příkaz zajistí předání parametrů hodnotou a předání řízení do těla funkce. V případě, že příkaz volání funkce obsahuje jiný počet nebo typy skutečných parametrů, než funkce očekává (tedy než je uvedeno v její hlavičce, a to i u vestavěných funkcí, včetně předání **null**, kde to není očekáváno), jedná se o chybu 4. Po dokončení provádění zavolané funkce je přiřazena návratová hodnota do proměnné *\$id* a běh programu pokračuje bezprostředně za příkazem volání právě provedené funkce. Je-li volána **void**-funkce, tak bude do *\$id* přiřazena hodnota **null**.

- *Volání funkce bez navracení hodnoty (tj. void-funkce):*

název_funkce (seznam_vstupních_parametrů) ;

Při volání **void**-funkce jsou *seznam_vstupních_parametrů* a sémantika příkazu analogické předchozímu příkazu. Bez přiřazení návratové hodnoty lze volat i funkci, která dle své definice hodnotu vrací, a ta je poté zahozena.

¹⁰Parametrem volání funkce není výraz. Jedná se o součást nepovinného bodovaného rozšíření projektu FUNEXP.

- *Příkaz návratu z funkce:*

return výraz ;

Příkaz může být použit v těle libovolné funkce nebo v hlavním těle programu. Jeho sémantika je následující: Dojde k vyhodnocení výrazu *výraz* (tj. získání návratové hodnoty), okamžitému ukončení provádění těla funkce (či celého programu) a návratu do místa volání, kam funkce vrátí vypočtenou návratovou hodnotu. Výjimkou je hlavní tělo programu, kde se provede vyhodnocení výrazu a ukončení běhu programu bez ovlivnění návratové hodnoty interpretu (tj. bezchybně interpretovaný, validní program bude i s provedením příkazu například `return 9;` vždy vracet 0). V případě těla **void**-funkce musí být výraz vynechán, jinak nastane chyba 6. Hlavní tělo programu ani uživatelem definovaná funkce bez návratové hodnoty nemusí příkaz **return** vůbec obsahovat. Chybí-li ve funkci vracející hodnotu výraz *výraz* u příkazu **return**, vede to na chybu 6. Pokud funkce s návratovou hodnotou vrací návratovou hodnotu neočekávaného typu dle její definice, dochází k chybě 4.

5 Výrazy

Výrazy jsou tvořeny termy, závorkami a aritmetickými, řetězcovými a relačními operátory.

V IFJ22 jsou prováděny typové kontroly a omezeny implicitní typové konverze u parametrů funkcí, kde jsou staticky uvedeny typy parametrů. Ve výrazech si operátory zajišťují kompatibilitu operandů pomocí implicitních konverzí (viz tabulka 1). Implementace konverze mezi číselnými typy a typem **string** jsou součástí rozšíření STRNUM, takže tyto implicitní konverze nebudou v základu testovány.

5.1 Aritmetické, řetězcové a relační operátory

Standardní binární operátory **+**, **-**, ***** značí sčítání, odčítání¹¹ a násobení. Jsou-li oba operandy typu **int**, je i výsledek typu **int**. Je-li jeden či oba operandy typu **float**, výsledek je typu **float**. Operátor **/** značí dělení dvou číselných operandů a výsledek je typu **float**¹². Pro provedení explicitního přetypování z **float** na **int** lze použít vestavěnou funkci **intval** (viz kapitola 6).

Řetězcový operátor **.** provádí se dvěma operandy typu **string** jejich konkatenci¹³

Pro operátor **===** platí: Pokud je první operand jiného typu než druhý operand, výsledek je **false** (takže `0.0 === "0.0"` bude vyhodnoceno jako **false**). Pokud jsou operandy stejného typu, tak se porovnají hodnoty daných operandů. Operátor **!==** je negací operátoru **===**.

Pro relační operátory **<**, **>**, **<=**, **>=** platí: Sémantika operátorů odpovídá jazyku PHP. Ze sémantiky jazyka PHP vycházejte i při porovnání s hodnotou **null**, kdy **<**, **>** dávají nepravdu a **<=**, **>=** dávají pravdu při porovnání s **" "**, **0** a **null**, jinak nepravdu. Na rozdíl od PHP se u řetězců porovnání vždy provádí lexikograficky¹⁴. Všechny typové nekompatibility

¹¹Číselné literály jsou sice nezáporné, ale výsledek výrazu přiřazený do proměnné již záporný být může.

¹²Celočíselné dělení tudíž neuvažujte.

¹³V případě, že je operand jiného typu, je implicitně převeden na typ **string** se sémantikou konverze popsané u vestavěné funkce **strval**.

¹⁴Neuvažujte tedy vlastnost PHP související s tzv. numerical strings.

ve výrazech jsou v IFJ22 (v konzistenci s PHP) řešeny implicitními konverzemi, z nichž jsou některé testovány jen v rámci rozšíření.

U řetězců se porovnání provádí lexikograficky. Bez rozšíření `BOOLTHERN` není s výsledkem porovnání možné dále pracovat a lze jej využít pouze u podmínek příkazů `if` a `while`.

5.2 Priorita operátorů

Prioritu operátorů lze explicitně upravit závorkováním podvýrazů. Následující tabulka udává priority operátorů (nahore nejvyšší):

Priorita	Operátory	Asociativita
1	<code>*</code> <code>/</code>	levá
2	<code>+</code> <code>-</code> <code>.</code>	levá
3	<code><</code> <code>></code> <code><=</code> <code>>=</code>	levá
4	<code>==</code> <code>!=</code>	levá

6 Vestavěné funkce

Překladač bude poskytovat některé základní vestavěné funkce, které bude možné využít v programech jazyka IFJ22. Pro generování kódu vestavěných funkcí lze výhodně využít specializovaných instrukcí jazyka IFJcode22.

Při použití špatného typu termu v parametrech následujících vestavěných funkcí dochází k chybě 4.

Vestavěné funkce pro načítání literálů a výpis termů:

- *Příkazy pro načítání hodnot:*

```
function reads() : ?string
function readi() : ?int
function readf() : ?float
```

Vestavěné funkce ze standardního vstupu načtou jeden řádek ukončený odřádkováním nebo koncem souboru (EOF). Funkce `reads` tento řetězec vrátí bez symbolu konce řádku (načítaný řetězec nepodporuje escape sekvence). V případě `readi` a `readf` jsou okolní bílé znaky ignorovány. Jakýkoli jiný nevhodný znak před či za samotným číslem je známkou špatného formátu a vede na návratovou hodnotu `null`. Funkce `readi` načítá a vrací celé číslo, `readf` desetinné číslo. V případě chybějící hodnoty na vstupu (např. načtení EOF) nebo jejího špatného formátu je vrácena hodnota `null`.

- *Příkaz pro výpis hodnot:*

```
function write ( term1 , term2 , ..., termn ) : void
```

Vestavěný příkaz má libovolný počet parametrů tvořených termy oddělenými čárkou.

¹⁴Asociativitu relačních operátorů není v základu třeba implementovat, případně viz rozšíření `BOOLTHERN`.

z/na	<i>int</i>	<i>float</i>	<i>bool</i>	<i>string</i>
<i>null</i>	0	0.0	false	" "
<i>int</i>	beze změny	pouze změna reprezentace	0 na false , jinak true	rozšíření STR- NUM
<i>float</i>	oříznutí desetinného rozvoje	beze změny	0.0 na false , jinak true	rozšíření STR- NUM
<i>bool</i> (rozšíření BOOL-THEN)	true na 1, false na 0	true na 1.0, false 0.0	beze změny	true na "1", false na " "
<i>string</i>	rozšíření STR- NUM	rozšíření STR- NUM	" " na false , jinak true	beze změny

Tabulka 1: Konverze datových typů (včetně rozšíření BOOLTHEN a STRNUM)

Sémantika příkazu je následující: Postupně zleva doprava prochází termy (podrobněji popsány v sekci 3.1) a vypisuje jejich hodnoty na standardní výstup ihned za sebe bez žádných oddělovačů dle typu v patřičném formátu. Za posledním termem se též nic nevypisuje! Hodnota termu typu **int** bude vytištěna pomocí `'%d'`¹⁵, hodnota termu typu **float** pak pomocí `'%a'`¹⁶. Hodnota **null** je pro tisk převedena na řetězec dle tabulky 1. Funkce **write** nemá návratovou hodnotu.

Vestavěné funkce pro konverzi typů: Konverze mezi číselnými a řetězcovými typy jsou součástí rozšíření STRNUM a nebudou v základních testech.

- **function floatval (term) : float** – Vráti hodnotu termu *term* konvertovanou na desetinné číslo (viz tabulka 1). Pro konverzi z celého čísla využijte odpovídající instrukci z IFJcode22.
- **function intval (term) : int** – Vráti hodnotu termu *term* konvertovanou na celé číslo (viz tabulka 1). Pro konverzi z desetinného čísla využijte odpovídající instrukci z IFJcode22.
- **function strval (term) : string** – Vráti hodnotu termu *term* konvertovanou na řetězec (viz tabulka 1). Tato funkce musí v základu (bez rozšíření STRNUM) podporovat jen term typu **string** a hodnotu **null**.

Vestavěné funkce pro práci s řetězci:

- **function strlen (string \$s) : int** – Vráti délku (počet znaků) řetězce \$s. Např. `strlen("x\nz")` vrátí 3.
- **function substring (string \$s, int \$i, int \$j) : ?string** – Vráti podřetězec zadaného řetězce *s*. Druhým parametrem *i* je dán index začátku požadovaného podřetězce (počítáno od nuly) a třetím parametrem *j* určuje index za posledním znakem podřetězce (též počítáno od nuly).

Funkce dále vrací hodnotu **null**, nastane-li některý z těchto případů:

¹⁵Formátovací řetězec standardní funkce **printf** jazyka C (standard C99 a novější).

¹⁶Formátovací řetězec **printf** jazyka C pro přesnou hexadecimální reprezentaci desetinného čísla, která bohužel není dostupná v PHP, ale odpovídající instrukce v IFJcode22 vypisuje ve správném formátu.

- $\$i < 0$
- $\$j < 0$
- $\$i > \j
- $\$i \geq \text{strlen}(\$s)$
- $\$j > \text{strlen}(\$s)$
- **function ord(string \$c) : int** – Vráti ordinální hodnotu (ASCII) prvního znaku v řetězci c . Je-li řetězec prázdný, vrací funkce 0.
- **function chr(int \$i) : string** – Vráti jednoznakový řetězec se znakem, jehož ASCII kód je zadán parametrem i . Hodnotu i mimo interval $[0; 255]$ řeší odpovídající instrukce IFJcode22.

7 Implementace tabulky symbolů

Tabulka symbolů bude implementována pomocí abstraktní datové struktury, která je ve variantě zadání pro daný tým označena identifikátory BVS a TRP, a to následovně:

BVS) Tabulku symbolů implementujte pomocí binárního vyhledávacího stromu.

TRP) Tabulku symbolů implementujte pomocí tabulky s rozptýlenými položkami.

Implementace tabulky symbolů bude uložena v souboru `symtable.c` (případně `symtable.h`). Více viz sekce 12.2.

8 Příklady

Tato kapitola uvádí tři jednoduché příklady řídicích programů v jazyce IFJ22.

8.1 Výpočet faktoriálu (iterativně)

```
<?php
declare(strict_types=1);
// Program 1: Vypocet faktorialu (iterativne)

// Hlavni telo programu
write("Zadejte cislo pro vypocet faktorialu\n");
$a = readi();

if ($a === null) {
    write("Chyba pri nacistani celeho cisla!\n");
} else {}

if ($a < 0) {
    write("Faktorial nelze spocitat\n");
} else {
    $vysl = 1;
    while ($a > 0) {
```

```

    $vysl = $vysl * $a;
    $a = $a - 1;
}
write("Vysledek je: ", $vysl, "\n");
}

```

8.2 Výpočet faktoriálu (rekurzivně)

```

<?php
declare(strict_types=1);
// Program 2: Vypocet faktorialu (rekurzivne)

// Hlavni telo programu
write("Zadejte cislo pro vypocet faktorialu: ");
$a = readi();

// Definice funkce pro vypocet hodnoty faktorialu
function factorial(int $n) : int {
    if ($n < 2) {
        $result = 1;
    } else {
        $decremented_n = $n - 1;
        $temp_result = factorial($decremented_n);
        $result = $n * $temp_result;
    }
    return $result;
}

if ($a !== null) {
    if ($a < 0) { // Pokracovani hlavniho tela programu
        write("Faktorial nelze spocitat\n");
    } else {
        $vysl = factorial($a);
        write("Vysledek je: ", $vysl, "\n");
    }
} else {
    write("Chyba pri nacistani celeho cisla!\n");
}

```

8.3 Práce s řetězcí a vestavěnými funkcemi

```

<?php
declare(strict_types=1);
/* Program 3: Prace s retezci a vestavenymi funkcemi */

$str1 = "Toto je nejaky text v programu jazyka IFJ22";
$str2 = $str1 . ", který jeste trochu obohatime";
write($str1, "\n", $str2, "\n");
write("Pozice retezce \"text\" v \"$str2: ", 15, "\n");
write("Zadejte serazenou posloupnost vseh malych pismen a-h, ");
$str1 = reads();
if ($str1 !== null) {

```

```

while ($str1 != "abcdefgh") {
    write("Spatne zadana posloupnost, zkuste znovu:\n");
    $str1 = reads();
}
} else { }
?>

```

9 Doporučení k testování

Programovací jazyk IFJ22 je schválně navržen tak, aby byl téměř kompatibilní s podmnožinou jazyka PHP 8¹⁷. Pokud si student není jistý, co by měl cílový kód přesně vykonat pro nějaký zdrojový kód jazyka IFJ22, může si to ověřit následovně. Z Moodle si stáhne ze souborů k projektu soubor `ifj22.php` obsahující kód, který doplňuje kompatibilitu IFJ22 s interpretem `php8.1` jazyka PHP 8 na serveru `merlin`. Soubor `ifj22.php` obsahuje definice vestavěných funkcí, které jsou součástí jazyka IFJ22, ale chybí v potřebné formě v jazyce PHP 8.

Váš program v jazyce IFJ22 uložený například v souboru `testPrg.php` pak lze interpretovat na serveru `merlin` například pomocí příkazu:

```
php8.1 ifj22.php testPrg.php < test.in > test.out
```

Tím lze jednoduše zkontrolovat, co by měl provést zadaný zdrojový kód, resp. vygenerovaný cílový kód. Je ale potřeba si uvědomit, že jazyk PHP 8 je nadmnožinou jazyka IFJ22, a tudíž může zpracovat i konstrukce, které nejsou v IFJ22 povolené (např. bohatší syntaxe a sémantika většiny příkazů, či dokonce zpětné nekompatibility). Výčet těchto odlišností bude uveden v Moodle IFJ a můžete jej diskutovat na fóru předmětu IFJ.

10 Cílový jazyk IFJcode22

Cílový jazyk IFJcode22 je mezikódem, který zahrnuje instrukce tříadresné (typicky se třemi argumenty) a zásobníkové (typicky bez parametrů a pracující s hodnotami na datovém zásobníku). Každá instrukce se skládá z operačního kódu (klíčové slovo s názvem instrukce), u kterého nezáleží na velikosti písmen (tj. case insensitive). Zbytek instrukcí tvoří operandy, u kterých na velikosti písmen záleží (tzv. case sensitive). Operandy oddělujeme libovolným nenulovým počtem mezer či tabulátorů. Odřádkování slouží pro oddělení jednotlivých instrukcí, takže na každém řádku je maximálně jedna instrukce a není povoleno jednu instrukci zapisovat na více řádků. Každý operand je tvořen proměnnou, konstantou nebo návěštím. V IFJcode22 jsou podporovány jednořádkové komentáře začínající mřížkou (`#`). Kód v jazyce IFJcode22 začíná úvodním řádkem s tečkou následovanou jménem jazyka:

```
.IFJcode22
```

¹⁷Online dokumentace k PHP: <https://www.php.net/manual/en/index.php>

10.1 Hodnotící interpret `ic22int`

Pro hodnocení a testování mezikódu v IFJcode22 je k dispozici interpret pro příkazovou řádku (`ic22int`):

```
ic22int prg.code < prg.in > prg.out
```

Chování interpretu lze upravovat pomocí přepínačů/parametrů příkazové řádky. Ná-povědu k nim získáte pomocí přepínače `--help`.

Proběhne-li interpretace bez chyb, vrací se návratová hodnota 0 (nula). Chybovým případům odpovídají následující návratové hodnoty:

- 50 - chybně zadané vstupní parametry na příkazovém řádku při spouštění interpretu.
- 51 - chyba při analýze (lexikální, syntaktická) vstupního kódu v IFJcode22.
- 52 - chyba při sémantických kontrolách vstupního kódu v IFJcode22.
- 53 - běhová chyba interpretace – špatné typy operandů.
- 54 - běhová chyba interpretace – přístup k neexistující proměnné (rámec existuje).
- 55 - běhová chyba interpretace – rámec neexistuje (např. čtení z prázdného zásobníku rámců).
- 56 - běhová chyba interpretace – chybějící hodnota (v proměnné, na datovém zásob-níku, nebo v zásobníku volání).
- 57 - běhová chyba interpretace – špatná hodnota operandu (např. dělení nulou, špatná návratová hodnota instrukce EXIT).
- 58 - běhová chyba interpretace – chybná práce s řetězcem.
- 60 - interní chyba interpretu tj. neovlivněná vstupním programem (např. chyba alo-kace paměti, chyba při otvírání souboru s řídicím programem atd.).

10.2 Paměťový model

Hodnoty během interpretace nejčastěji ukládáme do pojmenovaných proměnných, které jsou sdružovány do tzv. rámců, což jsou v podstatě slovníky proměnných s jejich hodnotami. IFJcode22 nabízí tři druhy rámců:

- globální, značíme GF (Global Frame), který je na začátku interpretace automaticky inicializován jako prázdný; slouží pro ukládání globálních proměnných;
- lokální, značíme LF (Local Frame), který je na začátku nedefinován a odkazuje na vrcholový/aktuální rámec na zásobníku rámců; slouží pro ukládání lokálních pro-měnných funkcí (zásobník rámců lze s výhodou využít při zanořeném či rekurzivním volání funkcí);
- dočasný, značíme TF (Temporary Frame), který slouží pro chystání nového nebo úklid starého rámcu (např. při volání nebo dokončování funkce), jenž může být přesu-nut na zásobník rámců a stát se aktuálním lokálním rámcem. Na začátku interpretace je dočasný rámec nedefinovaný.

K překrytým (dříve vloženým) lokálním rámcům v zásobníku rámců nelze přistoupit dříve, než vyjmeme později přidáné rámce.

Další možností pro ukládání nepojmenovaných hodnot je datový zásobník využívaný zásobníkovými instrukcemi.

10.3 Datové typy

Interpret IFJcode22 pracuje s typy operandů dynamicky, takže je typ proměnné (resp. paměťového místa) dán obsaženou hodnotou. Není-li řečeno jinak, jsou implicitní konverze zakázány. Interpret podporuje speciální hodnotu/typ `nil` a čtyři základní datové typy (`int`, `bool`, `float` a `string`), jejichž rozsahy i přesnosti jsou kompatibilní s jazykem IFJ22.

Zápis každé konstanty v IFJcode22 se skládá ze dvou částí oddělených zavináčem (znak `@`; bez bílých znaků), označení typu konstanty (`int`, `bool`, `float`, `string`, `nil`) a samotné konstanty (číslo, literál, `nil`). Např. `float@0x1.26666666666666p+0`, `bool@true`, `nil@nil` nebo `int@-5`.

Typ `int` reprezentuje 64-bitové celé číslo (rozsah C-long long `int`). Typ `bool` reprezentuje pravdivostní hodnotu (`true` nebo `false`). Typ `float` popisuje desetinné číslo (rozsah C-double) a v případě zápisu konstant používejte v jazyce C formátovací řetězec `'%a'` pro funkci `printf`. Literál pro typ `string` je v případě konstanty zapsán jako sekvence tisknutelných ASCII znaků (vyjma bílých znaků, mřížky (`#`) a zpětného lomítka (`\`)) a escape sekvencí, takže není ohraničen uvozovkami. Escape sekvence, která je nezbytná pro znaky s ASCII kódem 000-032, 035 a 092, je tvaru `\xyz`, kde `xyz` je dekadické číslo v rozmezí 000-255 složené právě ze tří číslic; např. konstanta

```
string@retezec\032s\032lomitkem\032\092\032a\010novym\035radkem
```

reprezentuje řetězec

```
retezec s lomitkem \ a
novym#radkem
```

Pokus o práci s neexistující proměnnou (čtení nebo zápis) vede na chybu 54. Pokus o čtení hodnoty neinicializované proměnné vede na chybu 56. Pokus o interpretaci instrukce s operandem nevhodných typů dle popisu dané instrukce vede na chybu 53.

10.4 Instrukční sada

U popisu instrukcí sázíme operační kód tučně a operandy zapisujeme pomocí neterminálních symbolů (případně číslovaných) v úhlových závorkách. Neterminál `<var>` značí proměnnou, `<symb>` konstantu nebo proměnnou, `<label>` značí návěští. Identifikátor proměnné se skládá ze dvou částí oddělených zavináčem (znak `@`; bez bílých znaků), označení rámce LF, TF nebo GF a samotného jména proměnné (sekvence libovolných alfanumerických a speciálních znaků bez bílých znaků začínající písmenem nebo speciálním znakem, kde speciální znaky jsou: `_`, `-`, `$`, `&`, `%`, `*`, `!`, `?`). Např. `GF@_x` značí proměnnou `_x` uloženou v globálním rámci.

Na zápis návěští se vztahují stejná pravidla jako na jméno proměnné (tj. část identifikátoru za zavináčem).

Instrukční sada nabízí instrukce pro práci s proměnnými v rámcích, různé skoky, operace s datovým zásobníkem, aritmetické, řetězcové, logické a relační operace, dále také konverzní, vstupně/výstupní a ladicí instrukce.

10.4.1 Práce s rámci, volání funkcí

MOVE $\langle var \rangle$ $\langle symb \rangle$	Přiřazení hodnoty do proměnné
Zkopíruje hodnotu $\langle symb \rangle$ do $\langle var \rangle$. Např. MOVE LF@par GF@var provede zkopírování hodnoty proměnné <i>var</i> v globálním rámci do proměnné <i>par</i> v lokálním rámci.	
CREATEFRAME	Vytvoř nový dočasný rámec
Vytvoří nový dočasný rámec a zahodí případný obsah původního dočasného rámce.	
PUSHFRAME	Přesun dočasného rámce na zásobník rámců
Přesuň TF na zásobník rámců. Rámec bude k dispozici přes LF a překryje původní rámec na zásobníku rámců. TF bude po provedení instrukce nedefinován a je třeba jej před dalším použitím vytvořit pomocí CREATEFRAME. Pokus o přístup k nedefinovanému rámci vede na chybu 55.	
POPFRAME	Přesun aktuálního rámce do dočasného
Přesuň vrcholový rámec LF ze zásobníku rámců do TF. Pokud žádný rámec v LF není k dispozici, dojde k chybě 55.	
DEFVAR $\langle var \rangle$	Definuj novou proměnnou v rámci
Definuje proměnnou v určeném rámci dle $\langle var \rangle$. Tato proměnná je zatím neinicilizovaná a bez určení typu, který bude určen až přiřazením nějaké hodnoty.	
CALL $\langle label \rangle$	Skok na návěští s podporou návratu
Uloží inkrementovanou aktuální pozici z interního čítače instrukcí do zásobníku volání a provede skok na zadané návěští (případnou přípravu rámce musí zajistit jiné instrukce).	
RETURN	Návrat na pozici uloženou instrukcí CALL
Vyjme pozici ze zásobníku volání a skočí na tuto pozici nastavením interního čítače instrukcí (úklid lokálních rámců musí zajistit jiné instrukce). Provedení instrukce při prázdném zásobníku volání vede na chybu 56.	

10.4.2 Práce s datovým zásobníkem

Operační kód zásobníkových instrukcí je zakončen písmenem „S“. Zásobníkové instrukce načítají chybějící operandy z datového zásobníku a výslednou hodnotu operace ukládají zpět na datový zásobník.

PUSHS $\langle symb \rangle$	Vlož hodnotu na vrchol datového zásobníku
Uloží hodnotu $\langle symb \rangle$ na datový zásobník.	
POPS $\langle var \rangle$	Vyjmi hodnotu z vrcholu datového zásobníku
Není-li zásobník prázdný, vyjme z něj hodnotu a uloží ji do proměnné $\langle var \rangle$, jinak dojde k chybě 56.	
CLEARs	Vymazání obsahu celého datového zásobníku
Pomocná instrukce, která smaže celý obsah datového zásobníku, aby neobsahoval zapomenuté hodnoty z předchozích výpočtů.	

10.4.3 Aritmetické, relační, booleovské a konverzní instrukce

V této sekci jsou popsány tříadresné i zásobníkové verze instrukcí pro klasické operace pro výpočet výrazu. Zásobníkové verze instrukcí z datového zásobníku vybírají operandy se vstupními hodnotami dle popisu tříadresné instrukce od konce (tj. typicky nejprve $\langle symb_2 \rangle$ a poté $\langle symb_1 \rangle$).

ADD $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$	Součet dvou číselných hodnot
Sečte $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ (musí být stejného číselného typu int nebo float) a výslednou hodnotu téhož typu uloží do proměnné $\langle var \rangle$.	
SUB $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$	Odečítání dvou číselných hodnot
Odečte $\langle symb_2 \rangle$ od $\langle symb_1 \rangle$ (musí být stejného číselného typu int nebo float) a výslednou hodnotu téhož typu uloží do proměnné $\langle var \rangle$.	
MUL $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$	Násobení dvou číselných hodnot
Vynásobí $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ (musí být stejného číselného typu int nebo float) a výslednou hodnotu téhož typu uloží do proměnné $\langle var \rangle$.	
DIV $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$	Dělení dvou desetinných hodnot
Podělí hodnotu ze $\langle symb_1 \rangle$ druhou hodnotou ze $\langle symb_2 \rangle$ (oba musí být typu float) a výsledek přiřadí do proměnné $\langle var \rangle$ (též typu float). Dělení nulou způsobí chybu 57.	
IDIV $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$	Dělení dvou celočíselných hodnot
Celočíselně podělí hodnotu ze $\langle symb_1 \rangle$ druhou hodnotou ze $\langle symb_2 \rangle$ (musí být oba typu int) a výsledek přiřadí do proměnné $\langle var \rangle$ typu int. Dělení nulou způsobí chybu 57.	
ADD/SUB/MUL/DIV/IDIV	Zásobníkové verze instrukcí ADD, SUB, MUL, DIV a IDIV
LT/GT/EQ $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$	Relační operátory menší, větší, rovno
Instrukce vyhodnotí relační operátor mezi $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ (stejného typu; int, bool, float nebo string) a do booleovské proměnné $\langle var \rangle$ zapíše false při neplatnosti nebo true v případě platnosti odpovídající relace. Řetězce jsou porovnávány lexikograficky a false je menší než true. Pro výpočet neostrých nerovností lze použít AND/OR/NOT. S operandem typu nil (druhý operand je libovolného typu) lze porovnávat pouze instrukcí EQ, jinak chyba 53.	
LTS/GTS/EQS	Zásobníková verze instrukcí LT/GT/EQ
AND/OR/NOT $\langle var \rangle \langle symb_1 \rangle \langle symb_2 \rangle$	Základní booleovské operátory
Aplikuje konjunkci (logické A)/disjunkci (logické NEBO) na operandy typu bool $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ nebo negaci na $\langle symb_1 \rangle$ (NOT má pouze 2 operandy) a výsledek typu bool zapíše do $\langle var \rangle$.	
ANDS/ORS/NOTS	Zásobníková verze instrukcí AND, OR a NOT
INT2FLOAT $\langle var \rangle \langle symb \rangle$	Převod celočíselné hodnoty na desetinnou
Převede celočíselnou hodnotu $\langle symb \rangle$ na desetinné číslo a uloží je do $\langle var \rangle$.	
FLOAT2INT $\langle var \rangle \langle symb \rangle$	Převod desetinné hodnoty na celočíselnou (oseknutí)
Převede desetinnou hodnotu $\langle symb \rangle$ na celočíselnou oseknutím desetinné části a uloží ji do $\langle var \rangle$.	

INT2CHAR $\langle var \rangle$ $\langle symb \rangle$	Převod celého čísla na znak
Číselná hodnota $\langle symb \rangle$ je dle ASCII převedena na znak, který tvoří jednoznakový řetězec přiřazený do $\langle var \rangle$. Je-li $\langle symb \rangle$ mimo interval $[0; 255]$, dojde k chybě 58.	
STR2INT $\langle var \rangle$ $\langle symb_1 \rangle$ $\langle symb_2 \rangle$	Ordinální hodnota znaku
Do $\langle var \rangle$ uloží ordinální hodnotu znaku (dle ASCII) v řetězci $\langle symb_1 \rangle$ na pozici $\langle symb_2 \rangle$ (indexováno od nuly). Indexace mimo daný řetězec vede na chybu 58.	
INT2FLOATS/FLOAT2INTS/INT2CHARS/STR2INTS	Zásobníkové verze konverzních instrukcí

10.4.4 Vstupně-výstupní instrukce

READ $\langle var \rangle$ $\langle type \rangle$	Načtení hodnoty ze standardního vstupu
Načte jednu hodnotu dle zadaného typu $\langle type \rangle \in \{\text{int, float, string, bool}\}$ (včetně případné konverze vstupní hodnoty float při zadaném typu int) a uloží tuto hodnotu do proměnné $\langle var \rangle$. Formát hodnot je kompatibilní s chováním vestavěných funkcí reads , readi a readf jazyka IFJ22.	
WRITE $\langle symb \rangle$	Výpis hodnoty na standardní výstup
Vypíše hodnotu $\langle symb \rangle$ na standardní výstup. Formát výpisu je kompatibilní s vestavěným příkazem write jazyka IFJ22 včetně výpisu desetinných čísel pomocí formátovacího řetězce "%a".	

10.4.5 Práce s řetězci

CONCAT $\langle var \rangle$ $\langle symb_1 \rangle$ $\langle symb_2 \rangle$	Konkatenace dvou řetězců
Do proměnné $\langle var \rangle$ uloží řetězec vzniklý konkatenací dvou řetězcových operandů $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ (jiné typy nejsou povoleny).	
STRLEN $\langle var \rangle$ $\langle symb \rangle$	Zjistí délku řetězce
Zjistí délku řetězce v $\langle symb \rangle$ a délka je uložena jako celé číslo do $\langle var \rangle$.	
GETCHAR $\langle var \rangle$ $\langle symb_1 \rangle$ $\langle symb_2 \rangle$	Vrať znak řetězce
Do $\langle var \rangle$ uloží řetězec z jednoho znaku v řetězci $\langle symb_1 \rangle$ na pozici $\langle symb_2 \rangle$ (indexováno celým číslem od nuly). Indexace mimo daný řetězec vede na chybu 58.	
SETCHAR $\langle var \rangle$ $\langle symb_1 \rangle$ $\langle symb_2 \rangle$	Změň znak řetězce
Zmodifikuje znak řetězce uloženého v proměnné $\langle var \rangle$ na pozici $\langle symb_1 \rangle$ (indexováno celočíselně od nuly) na znak v řetězci $\langle symb_2 \rangle$ (první znak, pokud obsahuje $\langle symb_2 \rangle$ více znaků). Výsledný řetězec je opět uložen do $\langle var \rangle$. Při indexaci mimo řetězec $\langle var \rangle$ nebo v případě prázdného řetězce v $\langle symb_2 \rangle$ dojde k chybě 58.	

10.4.6 Práce s typy

TYPE $\langle var \rangle$ $\langle symb \rangle$	Zjistí typ daného symbolu
Dynamicky zjistí typ symbolu $\langle symb \rangle$ a do $\langle var \rangle$ zapíše řetězec značící tento typ (int, bool, float, string nebo nil). Je-li $\langle symb \rangle$ neinicializovaná proměnná, označí její typ prázdným řetězcem.	

10.4.7 Instrukce pro řízení toku programu

Neterminál $\langle label \rangle$ označuje návěští, které slouží pro označení pozice v kódu IFJcode22. V případě skoku na neexistující návěští dojde k chybě 52.

LABEL $\langle label \rangle$

Definice návěští

Speciální instrukce označující pomocí návěští $\langle label \rangle$ důležitou pozici v kódu jako potenciální cíl libovolné skokové instrukce. Pokus o redefinici existujícího návěští je chybou 52.

JUMP $\langle label \rangle$

Nepodmíněný skok na návěští

Provede nepodmíněný skok na zadané návěští $\langle label \rangle$.

JUMPIFEQ $\langle label \rangle \langle symb_1 \rangle \langle symb_2 \rangle$

Podmíněný skok na návěští při rovnosti

Pokud jsou $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ stejného typu nebo je některý operand nil (jinak chyba 53) a zároveň se jejich hodnoty rovnají, tak provede skok na návěští $\langle label \rangle$.

JUMPIFNEQ $\langle label \rangle \langle symb_1 \rangle \langle symb_2 \rangle$

Podmíněný skok na návěští při nerovnosti

Jsou-li $\langle symb_1 \rangle$ a $\langle symb_2 \rangle$ stejného typu nebo je některý operand nil (jinak chyba 53), ale různé hodnoty, tak provede skok na návěští $\langle label \rangle$.

JUMPIFEQS/JUMPIFNEQS $\langle label \rangle$

Zásobníková verze JUMPIFEQ, JUMPIFNEQ

Zásobníkové skokové instrukce mají i jeden operand mimo datový zásobník, a to návěští $\langle label \rangle$, na které se případně provede skok.

EXIT $\langle symb \rangle$

Ukončení interpretace s návratovým kódem

Ukončí vykonávání programu a ukončí interpret s návratovým kódem $\langle symb \rangle$, kde $\langle symb \rangle$ je celé číslo v intervalu 0 až 49 (včetně). Nevalidní celočíselná hodnota $\langle symb \rangle$ vede na chybu 57.

10.4.8 Ladicí instrukce

BREAK

Výpis stavu interpretu na `stderr`

Na standardní chybový výstup (`stderr`) vypíše stav interpretu v danou chvíli (tj. během vykonávání této instrukce). Stav se mimo jiné skládá z pozice v kódu, výpisu globálního, aktuálního lokálního a dočasného rámce a počtu již vykonaných instrukcí.

DPRINT $\langle symb \rangle$

Výpis hodnoty na `stderr`

Vypíše zadanou hodnotu $\langle symb \rangle$ na standardní chybový výstup (`stderr`). Výpisy touto instrukcí bude možné vypnout pomocí volby interpretu (viz nápověda interpretu).

11 Pokyny ke způsobu vypracování a odevzdání

Tyto důležité informace nepodceňujte, neboť projekty bude částečně opravovat automat a nedodržení těchto pokynů povede k tomu, že automat daný projekt nebude schopen přeložit, zpracovat a ohodnotit, což může vést až ke ztrátě všech bodů z projektu!

11.1 Obecné informace


Za celý tým odevzdá projekt vedoucí. Všechny odevzdané soubory budou zkomprimovány programem ZIP, TAR+GZIP, nebo TAR+BZIP do jediného archivu, který se bude jmenovat `xlogin99.zip`, `xlogin99.tgz`, nebo `xlogin99.tbz`, kde místo zástupného


řetězce `xlogin99` použijte školní přihlašovací jméno **vedoucího** týmu. Archiv nesmí obsahovat adresářovou strukturu ani speciální či spustitelné soubory. Názvy všech souborů budou obsahovat pouze písmena¹⁸, číslice, tečku a podtržítko (ne mezery!).


Celý projekt je třeba odevzdat v daném termínu (viz výše). Pokud tomu tak nebude, je projekt považován za neodevzdaný. Stejně tak, pokud se bude jednat o plagiátorství jakéhokoliv druhu, je projekt hodnocený nula body, navíc v IFJ ani v IAL nebude udělen zápočet a bude zváženo zahájení disciplinárního řízení.


Vždy platí, že je třeba při řešení problémů aktivně a konstruktivně komunikovat nejen uvnitř týmu, ale občas i se cvičícím. Při komunikaci uvádějte login vedoucího a případně jméno týmu.


11.2 Dělení bodů

Odevzdaný archiv bude povinně obsahovat soubor **rozdeleni**, ve kterém zohledníte dělení bodů mezi jednotlivé členy týmu (i při požadavku na rovnoměrné dělení). Na každém řádku je uveden login jednoho člena týmu, bez mezery je následován dvojtečkou a po ní je bez mezery uveden požadovaný celočíselný počet procent bodů bez uvedení znaku %. Každý řádek (i poslední) je poté ihned ukončen jedním znakem <LF> (ASCII hodnota 10, tj. unixové ukončení řádku, ne windowsovské!). Obsah souboru bude vypadat například takto (() zastupuje unixové odřádkování):

xnovak01:30()

xnovak02:40()

xnovak03:30()

xnovak04:00()

Součet všech procent musí být roven 100. V případě chybného celkového součtu všech procent bude použito rovnoměrné rozdělení. Formát odevzdaného souboru musí být správný a obsahovat všechny registrované členy týmu (i ty hodnocené 0 %).

Vedoucí týmu je před odevzdáním projektu povinen celý tým informovat o rozdělení bodů. Každý člen týmu je navíc povinen rozdělení bodů zkontrolovat po odevzdání do StudIS a případně rozdělení bodů reklamovat u cvičícího ještě před obhajobou projektu.

12 Požadavky na řešení

Kromě požadavků na implementaci a dokumentaci obsahuje tato kapitola i několik rad pro zdárné řešení tohoto projektu a výčet rozšíření za prémiové body.

12.1 Závazné metody pro implementaci překladače

Projekt bude hodnocen pouze jako funkční celek, a nikoli jako soubor separátních, společně nekooperujících modulů. Při tvorbě lexikální analýzy využijete znalosti konečných automatů. Při konstrukci syntaktické analýzy založené na LL-gramatice (vše kromě výrazů) **povinně** využijte buď **metodu rekurzivního sestupu** (doporučeno), nebo prediktivní analýzu řízenou LL-tabulkou. Výrazy zpracujte pouze pomocí **precedenční syntaktické analýzy**. Vše bude probíráno na přednáškách v rámci předmětu IFJ. Implementace

¹⁸Po přejmenování změnou velkých písmen na malá musí být všechny názvy souborů stále unikátní.

bude provedena v **jazyce C**, čímž úmyslně omezujeme možnosti použití objektově orientované implementace. Návrh implementace překladače je zcela v režii řešitelských týmů. Není dovoleno spouštět další procesy a vytvářet nové či modifikovat existující soubory (ani v adresáři /tmp). Nedodržení těchto metod bude penalizováno značnou ztrátou bodů!

12.2 Implementace tabulky symbolů v souboru **syntable.c**

Implementaci tabulky symbolů (dle varianty zadání) proveďte dle přístupů probíraných v předmětu IAL a umístěte ji do souboru **syntable.c**. Pokud se rozhodnete o odlišný způsob implementace, vysvětlete v dokumentaci důvody, které vás k tomu vedly, a uveďte zdroje, ze kterých jste čerpali.

12.3 Textová část řešení

Součástí řešení bude dokumentace vypracovaná ve formátu PDF a uložená v jediném souboru **dokumentace.pdf**. Jakýkoliv jiný než předepsaný formát dokumentace bude ignorován, což povede ke ztrátě bodů za dokumentaci. Dokumentace bude vypracována v českém, slovenském nebo anglickém jazyce v rozsahu cca. 3-5 stran A4.

V dokumentaci popisujte návrh (části překladače a předávání informací mezi nimi), implementaci (použité datové struktury, tabulku symbolů, generování kódu), vývojový cyklus, způsob práce v týmu, speciální použité techniky a algoritmy a různé odchylky od přednášené látky či tradičních přístupů. Nezapomínejte také citovat literaturu a uvádět reference na čerpané zdroje včetně správné citace převzatých částí (obrázky, magické konstanty, vzorce). Nepopisujte záležitosti obecně známé či přednášené na naší fakultě.

Dokumentace musí povinně obsahovat (povinné tabulky a diagramy se nezapočítávají do doporučeného rozsahu):

- 1. strana: jména, příjmení a přihlašovací jména řešitelů (označení vedoucího) + údaje o rozdělení bodů, identifikaci vaší varianty zadání ve tvaru “Tým *login_vedoucího*, varianta X” a výčet identifikátorů implementovaných rozšíření.
- Rozdělení práce mezi členy týmu (uveďte kdo a jak se podílel na jednotlivých částech projektu; povinně zdůvodněte odchylky od rovnoměrného rozdělení bodů).
- Diagram konečného automatu, který specifikuje lexikální analyzátor.
- LL-gramatiku, LL-tabulku a precedenční tabulku, podle kterých jste implementovali váš syntaktický analyzátor.
- Stručný popis členění implementačního řešení včetně názvů souborů, kde jsou jednotlivé části včetně povinných implementovaných metod překladače k nalezení.

Dokumentace nesmí:

- obsahovat kopii zadání či text, obrázky¹⁹ nebo diagramy, které nejsou vaše původní (kopie z přednášek, sítě, WWW, ...).
- být založena pouze na výčtu a obecném popisu jednotlivých použitých metod (jde o váš vlastní přístup k řešení; a proto dokumentujte postup, kterým jste se při řešení ubírali; překážkách, se kterými jste se při řešení setkali; problémech, které jste řešili a jak jste je řešili; atd.)

¹⁹Vyjma obvyčejného loga fakulty na úvodní straně.

V rámci dokumentace bude rovněž vzat v úvahu stav kódu jako jeho čitelnost, srozumitelnost a dostatečné, ale nikoli přehnané komentáře.

12.4 Programová část řešení

Programová část řešení bude vypracována v jazyce C bez použití generátorů `lex/flex`, `yacc/bison` či jiných podobného ražení a musí být přeložitelná překladačem `gcc`. Při hodnocení budou projekty překládány na školním serveru `merlin`. 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, nebude projekt hodnocený. Ve sporných případech bude vždy za platný považován výsledek překladu a testování na serveru `merlin` bez použití jakýchkoliv dodatečných nastavení (proměnné prostředí, ...).

Součástí řešení bude soubor `Makefile` sloužící pro překlad projektu pomocí příkazu `make`. Pokud soubor pro sestavení cílového programu nebude obsažen nebo se na jeho základě nepodaří sestavit cílový program, nebude projekt hodnocený! Jméno cílového programu není rozhodující, bude přejmenován automaticky.

Binární soubor (přeložený překladač) v žádném případě do archívu nepřikládejte!

Úvod **všech** zdrojových textů musí obsahovat zakomentovaný název projektu, přihlašovací jména a jména studentů, kteří se na daném souboru skutečně autorsky podíleli.

Veškerá chybová hlášení vzniklá v průběhu činnosti překladače budou vždy vypisována na standardní chybový výstup. Veškeré texty tištěné řídicím programem budou vypisovány na standardní výstup, pokud není explicitně řečeno jinak. Kromě chybových/ladicích hlášení vypisovaných na standardní chybový výstup nebude generovaný mezikód přikazovat výpis žádných znaků či dokonce celých textů, které nejsou přímo předepsány řídicím programem. Základní testování bude probíhat pomocí automatu, který bude postupně vašim překladačem kompilovat sadu testovacích příkladů, kompilát interpretovat naším interpretem jazyka IFJcode22 a porovnávat produkované výstupy na standardní výstup s výstupy očekávanými. Pro porovnání výstupů bude použit program `diff` (viz `info diff`). Proto jediný neočekávaný znak, který bude při hodnotící interpretaci vámi vygenerovaného kódu svévolně vytisknut, povede k nevyhovujícímu hodnocení aktuálního výstupu, a tím snížení bodového hodnocení celého projektu.

12.5 Jak postupovat při řešení projektu

Při řešení je pochopitelně možné využít vlastní výpočetní techniku. Instalace překladače `gcc` není třeba, pokud máte již instalovaný jiný překladač jazyka C, avšak nesmíte v tomto překladači využívat vlastnosti, které `gcc` nepodporuje. Před použitím nějaké vyspělé konstrukce je dobré si ověřit, že jí disponuje i překladač `gcc` na serveru `merlin`. Po vypracování je též vhodné vše ověřit na serveru `Merlin`, aby při překladu a hodnocení projektu vše proběhlo bez problémů. V *Moodle* IFJ bude odkazován skript `is_it_ok.sh` na kontrolu většiny formálních požadavků odevzdávaného archívu, který doporučujeme využít.

Teoretické znalosti, potřebné pro vytvoření projektu, získáte během semestru na přednáškách, Moodle a diskuzním fóru IFJ. Postupuje-li Vaše realizace projektu rychleji než probírání témat na přednášce, doporučujeme využít samostudium (viz zveřejněné záznamy z minulých let a detailnější pokyny na Moodle IFJ). Je nezbytné, aby na řešení projektu

spolupracoval celý tým. Návrh překladáče, základních rozhraní a rozdělení práce lze vytvořit již v první čtvrtině semestru. Je dobré, když se celý tým domluví na pravidelných schůzkách a komunikačních kanálech, které bude během řešení projektu využívat (instant messaging, video konference, verzovací systém, štábní kulturu atd.).

Situaci, kdy je projekt ignorován částí týmu, lze řešit prostřednictvím souboru `rozdeleni` a extrémní případy řešte přímo se cvičícími co nejdříve. Je ale nutné, abyste si vzájemně (nespoléhejte pouze na vedoucího), nejlépe na pravidelných schůzkách týmu, ověřovali skutečný pokrok v práci na projektu a případně včas přerozdělili práci.

Maximální počet bodů získatelný na jednu osobu za programovou implementaci je **20** včetně bonusových bodů za rozšíření projektu.

Nenechávejte řešení projektu až na poslední týden. Projekt je tvořen z několika částí (např. lexikální analýza, syntaktická analýza, sémantická analýza, tabulka symbolů, generování mezikódu, dokumentace, testování!) a dimenzován tak, aby jednotlivé části bylo možno navrhnout a implementovat již v průběhu semestru na základě znalostí získaných na přednáškách předmětů IFJ a IAL a samostudiem na Moodle a diskuzním fóru předmětu IFJ.

12.6 Pokusné odevzdání

Pro zvýšení motivace studentů pro včasné vypracování projektu nabízíme koncept nepovinného pokusného odevzdání. Výměnou za pokusné odevzdání do uvedeného termínu (několik týdnů před finálním termínem) dostanete zpětnou vazbu v podobě procentuálního hodnocení aktuální kvality vašeho projektu.

Pokusné odevzdání bude relativně rychle vyhodnoceno automatickými testy a studentům zaslána informace o procentuální správnosti stěžejních částí pokusně odevzdaného projektu z hlediska části automatických testů (tj. nebude se jednat o finální hodnocení; proto nebudou sdělovány ani body). Výsledky nejsou nijak bodovány, a proto nebudou individuálně sdělovány žádné detaily k chybám v zaslaných projektech, jako je tomu u finálního termínu. Využití pokusného termínu není povinné, ale jeho nevyužití může být vzato v úvahu jako přitěžující okolnost v případě různých reklamací.

Formální požadavky na pokusné odevzdání jsou totožné s požadavky na finální termín a odevzdání se bude provádět do speciální aktivity „Projekt - Pokusné odevzdání“ předmětu IFJ. Není nutné zahrnout dokumentaci, která spolu s rozšířeními pokusně vyhodnocena nebude. Pokusně odevzdává nejvýše jeden člen týmu (nejlépe vedoucí), který se na zadání v rámci pokusného odevzdání registruje ve StudIS, odevzdává a následně obdrží jeho vyhodnocení, o kterém informuje zbytek týmu.

12.7 Registrovaná rozšíření

V případě implementace některých registrovaných rozšíření bude odevzdaný archiv obsahovat soubor **rozsiřeni**, ve kterém uvedete na každém řádku identifikátor jednoho implementovaného rozšíření (řádky jsou opět ukončeny znakem `<LF>`).

V průběhu řešení (do stanoveného termínu) bude postupně (případně i na váš popud) aktualizován ceník rozšíření a identifikátory rozšíření projektu (viz Moodle a diskuzní fórum k předmětu IFJ). V něm budou uvedena hodnocená rozšíření projektu, za která lze získat prémiové body. Cvičícím můžete během semestru zasílat návrhy na dosud neuvedená

rozšíření, která byste chtěli navíc implementovat. Cvičící rozhodnou o přijetí/nepřijetí rozšíření a hodnocení rozšíření dle jeho náročnosti včetně přiřazení unikátního identifikátoru. Body za implementovaná rozšíření se počítají do bodů za programovou implementaci, takže stále platí získatelné maximum 20 bodů.

12.7.1 Bodové hodnocení některých rozšíření jazyka IFJ22

Popis rozšíření vždy začíná jeho identifikátorem. Většina těchto rozšíření je založena na dalších vlastnostech jazyka PHP. Podrobnější informace lze získat ze specifikace jazyka² PHP. Do dokumentace je potřeba (kromě zkratky na úvodní stranu) také uvést, jak jsou implementovaná rozšíření řešena.

- **STRNUM**: Podpora implicitních konverzí čísel a řetězců včetně rozšíření vestavěných konverzních funkcí (+1,0 bodu).

function intval(*term*) : int a **function floatval(*term*) : float** – K základní implementaci jsou při konverzi z řetězce na desetinné číslo zahozeny všechny uvozující bílé znaky. Dále je načteno číslo dle definice *celočíslného* či *desetinného* literálu v kapitole 3.1 až po první nevyhovující znak (zbytek řetězce včetně tohoto znaku je ignorován) nebo konec řetězce. Není-li tímto způsobem načten ani jeden znak, vrátí funkce hodnotu 0 resp. 0.0. Je-li takto načtená posloupnost znaků nevyhovující definici číselného literálu, dojde k návratu hodnoty 0 resp. 0.0.

function strval(*term*) : string – Konverzi z celého, resp. desetinného čísla na řetězec provedte s výstupním formátem ' %d ', resp. ' %g ' ²⁰. U desetinných čísel budeme testovat přesnost na max. 4 desetinná místa. Čísla budou zadána v základním desítkovém tvaru.

- **BOOLTHEN**: Podpora typu **boolean**, booleovských hodnot **true** a **false**, booleovských výrazů včetně kulatých závorek a základních booleovských operátorů (**!**, **&&** a **||** včetně zkratového vyhodnocení), jejichž priorita a asociativita odpovídá jazyku PHP². Pravdivostní hodnoty lze porovnávat jen operátory **===** a **!==**. Dále podporujte výpisy hodnot typu **boolean**, implicitní konverze a přiřazování výsledku booleovského výrazu do proměnné. Dále podporujte rozšířený podmíněný příkaz **if-elseif-else** dle manuálu. Části **elseif** a **else** jsou tedy volitelné. Implementujte také vestavěnou funkci **function boolval(*term*) : boolean**, která vrátí hodnotu termu *term* konvertovanou na pravdivostní hodnotu dle tabulky 1 (+1,5 bodu).

- **CYCLES**: Překladač bude podporovat i cyklus **for**:

```
for ( $id1 = výraz1 ; výraz2 ; $id3 = výraz3 )
{
    sekvence_příkazů
}
```

kde *\$id₁* je proměnná (re)definovaná příkazem **for**. Libovolná ze tří částí hlavičky příkazu **for** může být i prázdná. Sémantika příkazu viz manuál². Podporujte i příkazy **break** a **continue** (+1,0 bodu). Příklad:

```
for ($i = 0; $i < 5; $i = $i + 1)
{
```

²⁰Formátovací řetězec standardní funkce **printf** jazyka C.


```

if ($i === 2)
{
    continue;
} else {}
write($i, "\n");
}

```

- FUNEXP: Volání funkce může být součástí výrazu, zároveň mohou být výrazy v parametrech volání funkce (+1,5 bodu).
- GLOBAL: Podporujte globální proměnné s využitím klíčového slova **global**. Kolize jmen proměnných řešte analogicky jako PHP (+0,5 bodu).

13 Opravy zadání

- 27. 9. 2022 – Oprava spodního limitu hexadecimálního zápisu escape sekvence v řetězci na `\x01`. Opravena návratová hodnota v sekci 4.1.1. Oprava překlepů.
- 28. 9. 2022 – Zpřehlednění popisu chyb ohledně návratových hodnot funkcí a úprava, že neprovedení **return** ve funkci s návratovou hodnotou způsobí chybu 4. Upřesnění vkládání bílých znaků na začátku a konci programu (před otevírací a za uzavírací značku nejsou povoleny žádné znaky ani bílé; viz sekce 4.1 a 4.3).