Otázky z diskuze IFJ-24

GENERAL

1. V zadání je napsáno:
   "Funkce *main* nemá definován žádný parametr ani návratovou hodnotu, jinak chyba 4."

   V jazyce IFJ24 tedy vypadá definice funkce main takto:
   pub fn main() void {
   statements
   }

   Chápu zadání správně, nebo správně nemá být deklarováno ani, že main je void funkcí?

   a. Vizte poslední odstavec na straně 6:

      „Funkce bez návratové hodnoty (návratový typ v hlavičce je **void**, tedy tzv. *void-funkce*) [...]“

      To, že funkce main nemá návratovou hodnotu, znamená, že je to funkce bez návratové hodnoty, tudíž typ v hlavičce je void, tudíž hlavička funkce main je skutečně pub fn main() void jak ostatně ukazují i dva příklady na straně 6...

2. Nenašly jsme v zadání omezení na počet vstupních parametrů funkci. Musíme považovat, že může být nekonečný nebo můžeme to nějakým způsobem omezit?

   . Teoreticky není důvod, aby byl počet parametrů funkce nějak omezen – gramatika musí umožňovat neomezený počet parametrů. Samozřejmě je jasné, že v praxi se nemůžeme takovému nekonečnu blížit.

3. Chtěl bych se zeptat jak udělat newline do terminálu v IFJcode24.

   . WRITE string@\010

4. Je možné v odůvodněných případech použít globální proměnné?

   . Ano, je to možné. V některých případech je použití globální proměnné spíše ku prospěchu věci a program se tím zpřehlední.

      . ZDOKUMENTOVAT

5. Odkaz na otázku**, je obsáhlá a při zkracování by mohlo dojít k ztrátě významu.**
6. Odkaz na otázku**, protože Křivka rád yappuje a je lepší si to od něj přečíst celý sám, než abych se to snažil zkrátit.**

LEX

1. Ohľadom ukončenia viacriadkových stringov. Podľa zadania je ukončenie \n pokiaľ sa ďalší riadok nezačína \\ ignorujúc biele znaky, ale na uloženie stringu do premennej musíme použiť funkciu ifj.string a neviem akým sposobom mám odlísiť ukočovaciu zátvorku funkcie od súčasti stringu. Môžeme predpokladať, že sa táto zátvorka bude vždy nachádzať ako prvý nebiely znak na novom riadku alebo ako sa k tomu postaviť?

    . Samozřejmě, že jakmile na řádku začne víceřádkový řetězec, už jeho součástí bude všechno až do konce řádku. Podívejte se do dokumentace jazyka Zig: https://ziglang.org/documentation/master/#Multiline-String-Literals , vidíte, že také mají středník až na dalším řádku. Zamyslete se tedy, jakým způsobem jste schopen rozlišit, že už skončilo načítání víceřádkového řetězce.
2. Jako klíčové slovo je uvedeno u8, ale datový typ je vždy []u8. Má překladač očekávat, že v závorkách nebo kolem nich nikdy nic nebude?
    . Je vhodné považovat **[**, **]** a **u8** za tři samostatné tokeny, přičemž gramatika nedovolí použít je jinak než v sekvenci []u8. Váš překladač by bílé znaky mezi těmito tokeny měl podporovat (obdobně jako Zig). Bílé znaky mezi závorkami a u8 nebudeme v základu testovat.
        . Dovolím si na otázku navázat. Měli bychom podobným způsobem zpracovávat i volitelnou předponu '?' nebo by měl v takovém případě syntaktický analyzátor zahlásit chybu?
            . Ano, pro prefix ? platí výše uvedené obdobně.
3. V zadání je, jak jste psal, že prolog se skládá z jednoho řádku
    const ifj = @import("ifj24.zig");
    Znamená to tedy, že tento string je pevně definovaný a to i co se týče bílých znaků mezi tokeny?
    . Existenci prologu budete ověřovat gramatickým pravidlem, které bude jistě obsahovat více než jeden terminál (token), nebylo by vhodné interpretovat prolog jen jako fixní řetězec *znaků*.
4. Chtěl bych se zeptat, pokud máme uvažovat maximální délku identifikátoru např. 256 znaků nebo máme raději přijít na dynamičtější řešení, přičemž nás bude omezovat pouze heap?
    . Nemá smysl to omezení explicitně dělat.
    . Mohu slíbit, že identifikátor delší jak 255 znaků testovat nebudeme, ale stejně vás to neosvobodí od dynamicky alokovaných řetězců.
5. Je korektní považovat identifikátor vestavěné funkce ve tvaru 'ifj.nazev_funkce' (kde mezi 'ifj', znakem '.' a 'nazev_funkce' může být libovolný počet bílých znaků) jako jeden typ tokenu?

- Je to v zásadě vaše implementační rozhodnutí, které pak popíšete v dokumentaci.
    - Definování funkce například: pub fn write() void by mělo být též chybné nebo ne? Vzhledem k tomu, že to vyloženě není totéž co ifj.write().
        - Identifikátor „write“ je zjevně jiný než identifikátor „ifj.write“

6. Je možné uložit víceřádkový řetězec do proměnné typu []u8 nebo je i v tomto případě nutné použít hned ifj.string()?
    - Z hlediska čehokoliv za lexikální analýzou se řetězcový literál chová úplně stejně nehledě na to, jestli byl původně jednořádkový, nebo víceřádkový;
7. V testu multiline.ifj:
   „//ukoncujici uvozovky ovlivnuji implicitni odsazeni vnitrnich radku retezce“.
   V zadání jsem nic o zvlášních vlastnostech uvozovek v multistringu nenašel, můžete to prosím nějak rozvést?
    - Na tom místě měl být komentář:
      „bile znaky za \ jsou soucasti retezce, posledni EOL ne, escape sekvence take ne“
8. Neobsahuje-li tělo funkce příkaz **return** a funkce by měla vracet hodnotu, dojde k chybě **6**.
   V tomto případě:

```
pub fn foo(param : i32) i32 {
  if (param == 42) {
    return 42;
  } else {}
}
```

   tělo funkce obsahuje příkaz return, ale volání s parametrem jiným než 42 return nedosáhne.
    - Nechceme po vás nějakou složitější statickou analýzu, která by ověřila, zda funkce ve všech větvích něco vrací (ale můžeš lol). Určitě by ale překladač měl být schopen zjistit 0 returnů (chyba 6).

SYNTAKTIKA

1. Pokud je typ „void“ použit v definici proměnné nebo parametru funkce, jde o syntaktickou, nebo sémantickou chybu? Zadání neuvádí „void“ jako datový typ, ale zároveň se tam o něm jako o typu v nějakém smyslu mluví.
    . Použití „void“ jinde než na místě návratového typu funkce je jistě chybou, kterou lze ověřit syntakticky, vhodnější je tedy hlásit syntaktickou chybu. Dá se to brát i za jinou chybu, ale musí to být odůvodněno v dokumentaci.
2. Dobrý den,

   V testu **„example2.zig“** na řádku 27 nalezneme takový kód:

   var result: i32 = -1;

   Pokud se podíváme do specifikace *„Číselné literály jsou sice nezáporné, ale výsledek výrazu přiřazený do proměnné již záporný být může“*, mělo by to být pravděpodobně interpretováno jako nelegitimní výraz.

   A ve specifikaci úlohy, v jednom z příkladů, nalezneme správný příklad takového výrazu:

   var result: i32 = 0 - 1;

   Můžeme druhý z nich považovat za původně zamýšlený výraz?

   Druhá otázka by zněla, pokud bychom v kódu viděli něco jako první výraz, které číslo chyby by nejlépe odpovídalo - chyba 7?

    . Máte pravdu, že to má být místo -1; správně dle IFJ24 jako 0 - 1;
    Co se týče čísla chyby, tak se zamyslete. Chyba 7 to není. Nejprve lex. analýza najde za tokenem = další token pro operátor -, a potom token celočíselný literál (tj. vše OK, takže lex. chyba to není). Pak narazíte na to, že vám něco chybí na zásobníku, když se bude mít redukovat na zásobníku handle "E - E", protože na zásobníku bude jen (dno zásobníku)"- E", což vede na syntaktickou chybu.
    Toto testovat nebudeme.
3. V základním zadání obsahují *if* a *while* pravdivostní výraz, který v sobě má vždy nějaký relační operátor. Na vyhodnocování výrazů musíme použít precedenční analýzu, ale v tomto případě by šlo také přímo do LL gramatiky zahrnout pravidlo „<exp> <rel-op> <exp>“
    . Ne. Udělej precedenční.
4. const x: ?i32 = 1;
   var y: i32 = x;

Co se má správně stát?

- ?i32 a i32 jsou dva různé typy, bude se chovat obdobně jako Zig.

5. Je u if povinný uzel else nebo se jedná o chybu číslo 2?
    - If bez else větve je součástí rozšíření BOOLTHEN.
        - V základním zadání je povinné else bráno jako zjednodušení odpovídající LL gramatiky. Nebudeme absenci else testovat v základu.
6. S jakým chybovým kódem by se měl ukončit program, pokud je identifikátor 'ifj' z prologu použit ve výrazu nebo při přiřazení do proměnné?

```
const ifj = @import("ifj24.zig");
  pub fn main() void {
    const a = ifj;
     ifj = 1 + 2;
  }
```

    - Prolog považujeme spíš za syntaktický požadavek bez sémantiky a nedá se říct, že by ze zadání úplně vyplývalo, že se samotné „ifj“ nedá použít jako identifikátor. Implementace záleží na vás, testovat to nebudeme.
7. Vím, že kdybych přiřazoval hodnotu do funkci tak to je chyba, ale je sémantická, či syntaktická? A taky přiřazovaní hodnoty do hodnoty?
    - Pokud zdrojový kód nelze vygenerovat vámi navrženou LL gramatikou, je to syntaktická chyba. Zkuste se blíže zamyslet, proč by tyto chyby neměli být syntaktické.
    - Taky se zkuste zamyslet, co by byla za chybu:
      some_function = 10;

SÉMANTIKA

1. chtěl bych se zeptat, zdali je v IFJ24 povoleno mít samostatný blok uvnitř jiného bloku? Např.:

```
pub fn myfun() void {
    const a = 5;
    {
        _ = a * a;
    }
}
```

   . V IFJ24 je nemusíte podporovat.

2. Může být příkaz return 'pouze' v bloku těla funkce, a nebo jestli může být i v jeho podblocích?
   . Nevidím důvod, proč by to mělo být omezené a může se vyskytovat i v příkazu větvení či v příkazu cyklu.
3. var nullable_prom : ?i32 = 10;
   var nenullable_prom : i32 = nullable_prom;

   Jedná se o sémantickou chybu č. 7 – nullable výraz nelze přiřadit do nenullable proměnné?

   Je pravda, že aritmetické operátory nepodporují žádné výrazy s datovým typem zahrnující null? Přítomnost by byla chyba 7.

   a. Ano, ano.

4. var a : i32 = 5; // definice proměnné typu i32
   a = 10.0; // rhs je konstantní výraz (v tomto případě literál) typu f64 s nulovou desetinnou částí -> bezeztrátová konverze na i32?

   const b : f64 = 5; // rhs je i32 literál -> konverze na f64?

   Je tento kód validní a mají se dané implicitní konverze provést nebo se jedná o error 7 - konkrétně o nekompatibilní typ výrazu při přiřazení/definici?

   a. Pokud to tak uděláte, rozhodně to bude *pěkné.* Ze zadání to však nevyplývá, nebudeme to v základu testovat.
5.
```
const a : ?i32 = 5;
if (a) |A| {
    const b : f64 = 1.2;
    if (A > b) {  // imp. konverze hodnoty A z i32 na f64?
        // something
```

} else {}
} else {}
Má A s typem i32 známou hodnotu při překladu? Šlo by tím ve výrazu A > b udělat konverze na f64? Nebo nastane chyba 7?

. Můžete se k tomu postavit oběma způsoby.

6. Je *var x: i32 = ifj.readi32();* sémanticky validní kód?

. Tento kód není sémanticky validní, porušuje typovou kompatibilitu.

7. *„Je-li jeden operand typu i32 a druhý f64 a celočíselný operand je zároveň literál, dojde k implicitní konverzi literálu na typ f64. Operátor / značí dělení (desetinné pro desetinné operandy, celočíselné pro celočíselné operandy).“*
Rozumím tomu tak, že ke konverzi by mělo dojít stejně jako u operací +,- a *.
Jak se má ale chovat překladač k situacím 42.2/8 a 12.0/8 ?

. V zadání je záměrně ta informace o implicitních konverzích uvedena jen pro +, - a *. Můžete si v Zigu vyzkoušet, že ani jeden z vámi uvedených výrazů nepřeloží. V prvním případě by to tedy byla jasná chyba nekompatibility typů operandů.

8. Pokud platí, že modifikovatelná proměnná musí být ve svém rozsahu platnosti změněna, jak se máme stavět k nenullable "proměnné" v případě větvení/cyklu s podmínkou zahrnující null? Dle zadání se jedná o nekonstasntní proměnnou.

. id_bez_null by se měla chovat jako konstanta a nutnost *změnit* ji tady samozřejmě není.

TABLE

1. Keďže program môže obsahovať neomezeně mnoho identifikátorov, ale neomezeně mnoho identifikátorov nemestíme do konečnej tabulky. Máme zvolit velkou tabulku nebo dynamicky?

. Obecnější řešení by bylo alokovat novou větší tabulku dynamicky a přelít všechna data z předešlé menší tabulky, ale jde to i staticky. Rozhodnutí zdůvodněte v dokumentaci.