ZADÁNÍ SEMESTRÁLNÍ PRÁCE
INTERPRET PODMNOŽINY JAZYKA LISP

Zadání

Naprogramujte v ANSI C přenositelnou[1] **konzolovou aplikaci**, která bude fungovat jako jednoduchý **interpret podmnožiny multiparadigmatického programovacího jazyka Lisp**. Vstupem aplikace bude seznam příkazů v jazyce ***Lisp***, buď uložený v textovém souboru, jehož název bude předaný jako parametr na příkazové řádce, nebo zadávaný interaktivně uživatelem z konzole. Výstupem je pak odpovídající posloupnost výsledků vyhodnocení každého zadaného výrazu.

Program se bude spouštět příkazem `lisp.exe` [⟨*vstupní-soubor*⟩] [`-v`]. Symbol ⟨*vstupní-soubor*⟩ zastupuje nepovinný parametr – název vstupního textového souboru se seznamem výrazů v jazyce Lisp (resp. jeho podmnožině). Není-li první parametr uveden, program bude fungovat v tzv. *interaktivním módu*, kdy se příkazy budou provádět přímo zadáním příslušného výrazu z konzole do interpretu a každý výraz bude okamžitě vyhodnocen (a výsledek tohoto vyhodnocení vypsán na konzoli).

V případě spuštění s parametrem určujícím vstupní soubor – tedy v tzv. *dávkovém módu* může být na příkazové řádce uveden ještě také přepínač '`-v`' (= verbose): Při jeho uvedení se při dávkovém zpracování výsledek vyhodnocení každého zpracovávaného výrazu současně vypíše do konzole (tedy jako při činnosti v interaktivním módu). Pokud přepínač '`-v`' není uveden, bude interpret „potichu" a každý zpracovávaný výraz pouze vyhodnotí, ale nebude nic vypisovat (pochopitelně kromě toho, co se vypisovat má jako následek vykonání funkce '`print`').

Vámi vyvinutý program tedy bude vykonávat následující činnosti:

1. Při spuštění bez parametru bude čekat na vstup od uživatele. Zadaný výraz v jazyce Lisp vyhodnotí, vyhodnocení vypíše a bude vyčkávat na další vstup, dokud nebude uveden výraz '`(quit)`', kterým se činnost interpretu ukončí.

2. Při spuštění s parametrem načte zadaný vstupní textový soubor se zdrojovým kódem v jazyce Lisp, každý výraz v něm uvedený okamžitě vyhodnotí a pokud byl na příkazové řádce při spuštění interpretu použit i přepínač '`-v`', výsledek vyhodnocení vypíše do konzole. Nebyl-li přepínač '`-v`' použit, vypisuje interpret do konzole pouze argumenty funkce '`print`' a případná chybová hlášení, jinak nic. Po zpracování posledního výrazu dojde k ukončení interpretu. Proto nemusí být jako poslední příkaz uveden ve zdrojovém textu výraz '`(quit)`'. Na jedné řádce v souboru však může být uvedeno více samostatných výrazů a program je musí být schopen správně zpracovat.

Během testování může být Váš program spuštěn například takto (v operačním systému Windows):

```
X:\>lisp.exe "E:\My Work\Test\test01.lisp" ↵
```

nebo takto:

```
X:\>lisp.exe src\test02.lisp -v ↵
```

nebo pouze takto (pro spuštění interaktivního režimu):

[1] Je třeba, aby bylo možné Váš program přeložit a spustit na PC s operačním prostředím Win32/64 (tj. operační systémy Microsoft Windows NT/2000/XP/Vista/7/8/10/11) a s běžnými distribucemi Linuxu (např. Ubuntu, Debian, Red Hat, atp.). Server, na který budete Vaši práci odevzdávat a který ji otestuje, má nainstalovaný operační systém Debian GNU/Linux 11 (bullseye) s jádrem verze 5.10.0-32-amd64 a s překladačem gcc 10.2.1-6.

```
X:\>lisp.exe ↵
```

Spuštění v UNIXových operačních systémech (GNU/Linux, FreeBSD, macOS, atp.) může vypadat např. takto:

```
$./lisp.exe /home/user/work/test03.lisp ↵
```

Uvažujte všechny možné specifikace (uvedení úplné cesty, relativní cesty, atp.) vstupního souboru, které jsou přípustné v daném operačním systému. Pokud parametr uvedený na příkazové řádce nebude korektní jméno existujícího souboru, který lze otevřít pro čtení, vypište chybové hlášení a stručný návod k použití programu (v angličtině). Návratová hodnota funkce `int main(...)` bude v tomto případě `1`.

Hotovou práci odevzdejte v jediném archivu typu ZIP prostřednictvím automatického odevzdávacího a validačního systému. Postupujte podle instrukcí uvedených na webu předmětu. Archiv nechť obsahuje všechny zdrojové soubory potřebné k přeložení programu, **makefile** pro Windows i Linux (pro překlad v Linuxu připravte soubor pojmenovaný `makefile` a pro Windows `makefile.win`) a dokumentaci ve formátu PDF vytvořenou v typografickém systému TEX, resp. LATEX. **Bude-li některá z částí chybět, validační systém Vaši práci odmítne.**

Podrobné informace k odevzdání práce najdete na webové stránce předmětu Programování v jazyce C na adrese URL `https://www.kiv.zcu.cz/studies/predmety/pc/index.php#work`.

Popis činnosti programu

Program je interpretem podmnožiny multiparadigmatického programovacího jazyka ***Lisp***, jehož syntax je významně ovlivněna lambda kalkulem, viz např. popis na `https://en.wikipedia.org/wiki/Lisp_(programming_language)`. Interpret zpracovává postupně zadané výrazy jazyka Lisp (resp. jeho níže specifikované podmnožiny), po zadání (ať už z konzole nebo načtením ze souboru) výraz vyhodnotí a v případě spuštění v interaktivním režimu výsledek vyhodnocení okamžitě vypíše do konzole (totéž udělá i v dávkovém režimu při zadání přepínače '`-v`').

Minimální množina operátorů a funkcí, které tento interpret musí zvládnout zpracovat, aby úspěšně prošel validátorem, je uvedena v tabulce 1 na str. 3 (z vlastní iniciativy můžete samozřejmě naprogramovat i další operátory/funkce Lispu, aby se váš program více přiblížil skutečnému plnohodnotnému interpretu jazyka Lisp).

Kdekoliv ve zdrojovém kódu se může vyskytnout **komentář**. Komentáře interpret ignoruje. Komentář začíná znakem '`;`' (středník) a končí s koncem řádky, tj. tyto komentáře nelze vkládat do vnitřku příkazů.

Specifikace vyhodnocovaných výrazů

Výrazem může být pouze:

1. **konstanta** — Celočíselná hodnota. Vyhodnocení konstanty vrací přímo danou celočíselnou hodnotu, která byla konstantě přiřazena (vyhodnocení tzv. „samo na sebe“). Jiné konstanty než celočíselné, tedy např. znakové, řetězcové nebo reálná čísla, interpret nemusí zpracovávat (ale může, pokud by to programátora bavilo).

2. **proměnná** — Název, pod kterým je uložena nějaká celočíselná hodnota nebo seznam. Její vyhodnocení vrací uloženou celočíselnou hodnotu nebo obsah seznamu (vkládání hodnoty proměnné se provádí funkcí '`set`', tj. např. '`(set a 1)`' nebo '`(set my-list '(1 2 3 4 5))`'. Deklarace je dynamická, tj. proměnná vznikne okamžikem, kdy je do ní přiřazena hodnota. Jiné proměnné než seznamy a celočíselné, tedy např. znakové, řetězcové nebo reálná čísla, interpret nemusí zpracovávat (ale může).

3. **seznam** — Seznam je dán výčtem prvků seznamu, uzavřeným v kulatých závorkách. Prvkem seznamu může být konstanta, proměnná, seznam nebo operátor či název funkce. Operátor či název funkce (pokud se v seznamu vyskytuje) je vždy uveden na prvním místě a seznam se tím stává vykonatelný ve smyslu volání dané funkce s argumenty danými následujícími prvky seznamu nebo aplikace operátoru s operandy danými následujícími prvky seznamu. Seznamy lze libovolně vnořovat, tj. např. '`(* (+ 1 2) (+ 3 4))`'.

   Interpret bude podporovat (tedy schopen vykonávat) minimálně operátory a funkce dané tabulkou 1.

Podpora dalších operátorů či funkcí záleží na rozhodnutí (a píli) programátora. Veškeré názvy a textové konstanty jsou ukládány do vnitřních struktur interpretu vždy s **velkými písmeny**, tj. interpret je **case-insensitive** (tj. nerozlišuje velká a malá písmena). Při výpisu vyhodnocení výrazu pak vždy používá velká písmena, tj. např. výraz '`'(a b c d)`' bude při vyhodnocení v interaktivním režimu vypsán jako '`(A B C D)`'.

Tabulka 1: Minimální množina podporovaných operátorů a funkcí.

Operátor / **Funkce**	**Význam**	**Příklad**	**Výsledek**
	Aritmetické operátory		
`+`	Sečte operandy zleva doprava (ne že by na tom v tomto případě záleželo) a vrátí celočíselný výsledek.	`(+ 1 2 5)`	`8`
`-`	Odečte operandy zleva doprava a vrátí celočíselný výsledek.	`(- 5 3)` `(- 20 10 5 2)`	`2` `3`
`*`	Vynásobí operandy zleva doprava (dtto) a vrátí celočíselný výsledek.	`(* 6 2)` `(* 5 4 3 2)`	`12` `120`

`/`	Vydělí sebou operandy zleva doprava a vrátí celočíselný výsledek. Prováděno bude vždy jen celočíselné dělení.	`(/ 8 2)`
		`(/ 3 2)`
		`(/ 36 2 3 2)`
			`4`
			`1`
			`3`

`inc`	Zvýší obsah celočíselné proměnné, která je prvním argumentem, o hodnotu druhého argumentu. Uvedení více než 2 argumentů je syntaktická chyba.	`(set a 1)` `(inc a 1)`	 `2`
`dec`	Sníží obsah celočíselné proměnné, která je prvním argumentem, o hodnotu druhého argumentu. Uvedení více než 2 argumentů je syntaktická chyba.	`(set a 5)` `(dec a 2)`	 `3`
	Relační operátory		

`=`	Porovná všechny operandy na shodu a výsledkem je `T` (true) nebo `NIL`. Výsledek je dán logickým součinem všech porovnání.	`(= 2 2)`
		`(= 3 1)`
		`(= 7 7 7)`
			`T`
			`NIL`
			`T`

`/=`	Porovná všechny operandy na neshodu a výsledkem je `T` nebo `NIL`. Výsledek je dán logickým součinem všech porovnání.	`(/= 3 3)`
		`(/= 3 2 2)`
		`(/= 3 2 1)`
			`NIL`
			`NIL`
			`T`

`<`	Porovná všechny operandy operátorem $<$ (menší než) a vrátí výsledek `T` nebo `NIL`. Výsledek je dán logickým součinem všech porovnání.	`(< 3 1 4)` `(< 1 2 3)`	`NIL` `T`
`>`	Porovná všechny operandy operátorem $>$ (větší než) a vrátí výsledek `T` nebo `NIL`. Výsledek je dán logickým součinem všech porovnání.	`(> 3 2 1)` `(> 4 1 5)`	`T` `NIL`
`<=`	Porovná všechny operandy operátorem $\leq$ (menší než nebo rovno) a vrátí výsledek `T` nebo `NIL`. Výsledek je dán logickým součinem všech porovnání.	`(<= 2 2 2 4)` `(<= 3 2)`	`T` `NIL`
`>=`	Porovná všechny operandy operátorem $\geq$ (větší než nebo rovno) a vrátí výsledek `T` nebo `NIL`. Výsledek je dán logickým součinem všech porovnání.	`(>= 2 2 2 4)` `(>= 3 2)`	`NIL` `T`
`max`	Porovná všechny operandy a nalezne mezi nimi **maximum**, které vrátí jako výsledek.	`(max 1 5 3 7)`	`7`

`min`	Porovná všechny operandy a nalezne mezi nimi **minimum**, které vrátí jako výsledek.	`(min 1 5 3 7)`	`1`
Funkce			
`quote`	Argument nebude vyhodnocen.	`(quote A)` `(quote (a b))`	`A` `(A B)`
`'`	Stejné chování jako funkce `quote` (jde vlastně o její zjednodušený, zrychlený zápis).	`'(a b 2)`	`(A B 2)`
`set`	Uloží do proměnné dané prvním argumentem hodnotu danou druhým argumentem. Funkce vrací ukládanou hodnotu. Uložená hodnota může být použita v dalších výrazech.	`(set a 4)` `(set ml '(1 2 3))`	`4` `(1 2 3)`
`list`	Argumenty funkce budou vráceny jako seznam.	`(list 1 2 3)` `(list 1 (list 2 3))`	`(1 2 3)` `(1 (2 3))`
`atom`	Testuje, zda je argument atomický, tj. je to celočíselná proměnná nebo konstanta, nebo – jednoduše řečeno – cokoliv jiného než seznam.	`(atom '(1 2 3))`	`NIL`
`car`	Vrátí první prvek (tzv. *hlavu*) seznamu předaného jako argument.	`(car '(1 2 3 4))`	`1`
`cdr`	Vrátí seznam předaný jako argument bez prvního prvku.	`(cdr '(1 2 3 4))`	`(2 3 4)`
`nth`	Vrátí n-tý prvek seznamu předaného jako druhý argument. Prvním argumentem je pořadí požadovaného prvku v seznamu, n. Prvky jsou indexované od nuly.	`(nth 2 '(1 2 3 4))`	`3`
`length`	Vrátí počet prvků seznamu předaného jako jediný argument.	`(length '(1 2 3 4))`	`4`
Řídicí konstrukce			

`if`	Vyhodnotí výraz následující za operátorem `if` a pokud je `T`, vyhodnotí třetí prvek seznamu, pokud ne, vyhodnotí čtvrtý prvek seznamu. Není-li čtvrtý prvek seznamu („else“ větev) uveden, neudělá nic.	`(if (> a 0) 1 2)`
		`(if (<= a 10)`
		`  (print a))`
			`1`
			`1`

`while`	Vyhodnotí výraz následující za operátorem `while` a dokud má hodnotu `T` (tedy je splněn), vyhodnocuje cyklicky všechny zbývající výrazy seznamu.	`(while (> a 0)`
		`  (print a)`
		`  (dec a 1))`
			`3`
			`2`
			`1`

`brk`	Ukončí provádění cyklu.	`(if (= a 2) (brk))`	–
Funkce interakce s prostředím			

`print`	Vytiskne obsah svého jediného argumentu – může to být konstanta, celočíselná proměnná a samozřejmě i seznam (neboť seznam může být obsahem proměnné, viz výše). Za výpisem hodnoty následuje znak (sekvence znaků) konce řádky CR LF (řešte kódem '`\n`'). Uvedení více než 1 argumentu je syntaktická chyba.	`(print 5)`
		`(print a)`
		`(print '(1 2 3))`
			`5`
			`4`
			`(1 2 3)`

`quit`	Ukončí program (a tedy i činnost interpretu).	`(quit)`	–

Tato podmnožina podporovaných syntaktických konstrukcí jazyka Lisp by měla dovolit naprogramovat jednoduché algoritmy, jako např. řazení pole technikou *Bubble Sort*.

Specifikace výstupu programu

V interaktivním režimu bude interpret vždy vypisovat výsledek vyhodnocení právě zadaného výrazu. Zadá-li tedy uživatel výraz '`(* (+ 1 2) (+ 3 4))` ↵', interpret vypíše na další řádce výsledek vyhodnocení tohoto výrazu, tedy '`21` CR LF'.

V případě funkce '`print`' je výsledkem vyhodnocení to, co bude vypsáno do konzole, tj. je-li funkce '`print`' vykonána v interaktivním režimu, vypíše do konzole obsah svého argumentu:

```
[1]> (print '(a b c))
(A B C)
```

Při načtení seznamu příkazů ze souboru (dávkový režim) se vyhodnocení právě zpracovávaného výrazu nevypisuje. Jediný výstup na konzoli probíhá následkem volání funkce '`print`', a pak samozřejmě chybové hlášení interpretu, pokud nastane nějaká chyba.

Chybová hlášení

Program vypíše chybové hlášení tehdy, nastane-li ve vyhodnocovaném výrazu některá jedna z následujících chyb:

- První prvek seznamu v roli příkazu není operátor nebo funkce. Výjimku tvoří argument funkce '`quote`' (resp. její zkrácené varianty '`'`', apostrof) a '`list`', které brání vyhodnocení (vykonání) seznamu.
- Jeden nebo více ostatních prvků (tedy kromě prvního) seznamu není konstanta, seznam, existující proměnná a nebo není uvedena jako argument funkce '`quote`'.
- Použití prázdného seznamu (tj. hodnoty '`NIL`') jako operandu. Hodnota '`NIL`' není číslo.

Návratová hodnota funkce `int main(...)` bude v tomto případě (tedy při výskytu syntaktické chyby) `2`.

Pokud nastane závažný chybový stav, který nedovoluje interpretu pokračovat v činnosti, oznamuje to uživateli srozumitelným chybovým hlášením v anglickém jazyce vypsaným na konzoli (do standardního proudu `stderr`) a následně vynuceným ukončením programu s předáním návratové hodnoty podle tabulky 2. Jiným než výše popsaným způsobem interpret během své činnosti s uživatelem neinteraguje.

Tabulka 2: Návratové kódy programu v případě chyby.

Popis chybového stavu	**Návratová hodnota funkce `int main()`**
`ERR_NO_ERROR`: Bez chyby/korektní ukončení činnosti interpretu.	`0`
`ERR_INVALID_INPUT_FILE`: Neplatný název vstupního souboru, nebo soubor neexistuje či nelze otevřít pro čtení.	`1`
`ERR_SYNTAX_ERROR`: **Syntaktická chyba** ve zdrojovém textu programu v Lispu. Interpret narazil např. na příkaz, který není uveden v tab. 1 nebo u příkazu chybí parametr(y) apod.	`2`
`ERR_FILE_ACCESS_FAILURE`: Soubor (vstupní či výstupní) je nepřístupný, nelze s ním pracovat (protože je např. otevřen ve výhradním režimu jiným procesem).	`3`
`ERR_OUT_OF_MEMORY`: Interpretu došla operační paměť. To by se stát nemělo, ale pokud ano, bude program reagovat tímto návratovým kódem.	`4`
Ostatním chybovým stavům můžete přiřadit návratové kódy dle svého uvážení.	

Užitečné techniky a odkazy

Uvedené techniky je možné (ale nikoliv nezbytně nutné) využít při řešení úlohy. Protože se jedná o postupy víceméně standardní, lze k nim nalézt velké množství dokumentace:

1. **S-expression** – `http://en.wikipedia.org/wiki/S-expression`

2. **Syntaktická analýza rekurzivním sestupem**
   – `https://en.wikipedia.org/wiki/Recursive_descent_parser`
3. **Interpret Lispu**, k otestování zda váš interpret vykonává příkazy Lispu správně
   – `https://www.tutorialspoint.com/execute_lisp_online.php`

Řešení úlohy je zcela ve vaší kompetenci – zvolte takové algoritmy a techniky, které podle vás nejlépe povedou k cíli…

Příloha 1 – Ukázka interpretace

```
[1]> (+ 4 6 (* 2 6 7 (+ 5 2)))
598
[2]> ()
NIL
[3]> (set 'a 10)
10
[4]> (+ 3 a)
13
[5]> (= 1 (- 2 1))
T
[6]> (/= 2 (/ 4 2))
NIL
[7]> '(a b (c d))
(A B (C D))
[8]> (car (list 1 2 3))
1
[9]> (cdr (list 2 3 5))
(3 5)
[10]> (car (list (list 1 2)))
(1 2)
[11]> 5
5
[12]> a
10
[13]> (quit)
Bye.
```

Příloha 2 – Příklad zdrojového souboru

Uvedený obsah souboru odpovídá seznamu příkazů v příloze 1 (jen chybí příkaz `(quit)`).

```
(+ 4 6 (* 2 6 7 (+ 5 2)))
()
(set 'a 10) (+ 3 a)
(= 1 (- 2 1))
(/= 2 (/ 4 2))
'(a b (c d))
(car (list 1 2 3))
(cdr (list 2 3 5))
(car (list (list 1 2)))
5 a
```

Příloha 3 – Zdrojový kód programu pro řazení seznamu technikou *Bubble Sort*

Níže uvedený zdrojový kód by měl interpret zvládnout korektně vykonat. Výsledkem je výpis seřazeného seznamu do konzole.

```
(set arr '(5 1 7 3 2 6 4 9 8))

(set swapped T)
(set len (length arr))
(while swapped
     (set i 1)
     (set swapped nil)
     (while (< i len)
          (set num1 (nth (- i 1) arr))
          (set num2 (nth i arr))
          (if (> num1 num2) ; swap arr[i] <--> arr[i - 1]
               ((set (nth (- i 1) arr) num2)
                (set (nth i arr) num1)
                (set swapped T))
          )
          (inc i 1)
     )
)

(print arr)
```