STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Programovací jazyk Kobeři-C

Filip Peterek
Moravskoslezský kraj, Kobeřice, 2018

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Programovací jazyk Kobeři-C

Programming language Köberwi-C

Autoři: Filip Peterek
Škola: Střední škola průmyslová a umělecká, Opava, Praskova 399/8, 746 01 Opava
Kraj: Moravskoslezský kraj
Konzultant: Mgr. Marek Lučný

Kobeřice 2018

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Kobeřicích dne 26. 3 2018 ……

Filip Peterek

Poděkování

Rád bych poděkoval své rodině, která mě vždy vedla k informatice a k tomu, abych byl ambiciózní a snažil se být ve všem nejlepší. Také děkuji Zuzce Matyáškové, která mě při vývoji jazyka Kobeři-C podporovala.

Anotace

Výsledkem projektu je funkční imperativní a objektově orientovaný programovací jazyk Kobeři-C a pro něj vytvořený transkompilátor, který překládá zdrojový kód v jazyce Kobeři-C do jazyka C. Kompilátor je implementován v jazyce C++. Použity jsou pouze standardní knihovny jazyka C++, jako např. iostream, vector, map… Nejsou použity žádné externí knihovny. Kompilátor je tvořen čtyřmi částmi - tokenizerem, parserem, analyzerem a translatorem. Aby jazyk Kobeři-C nevznikl zbytečně, přináší do jazyka C několik nových funkcí známých z jiných jazyků, jako jsou například třídy nebo přetěžování funkcí. Zdrojový kód v jazyce Kobeři-C je díky těmto funkcím v mnoha případech kratší a přehlednější než kód v jazyce C.

Klíčová slova

Kompilátor; transkompilace; programovací jazyk

Annotation

The result of my project is an imperative and object oriented programming language Köberwi-C along with a working transcompiler, which translates Köberwi-C code into the C programming language. The compiler is implemented in the C++ programming language and only uses libraries from the C++ standard library (e.g. iostream, vector, map...). No third-party libraries are used. The compiler consists of four parts: Tokenizer, Parser, Analyzer and Translator. The Köberwi-C programming language introduces features, which have been present in other languages for years, but are missing in C. These features include classes, function overloading or type deduction. Thanks to these features, code written in Köberwi-C tends to be shorter and more readable than code written in C.

Keywords

Compiler; transcompilation; programming language

Obsah

1 Úvod

K tvorbě vlastního programovacího jazyka mě vedly vysoké ambice. Vždy chci být první, vždy chci být nejlepší. Mým prvním nápadem na závěrečný projekt byl operační systém. Tento projekt by ale byl příliš složitý, obsahoval by spoustu low-level komunikace s hardwarem a neměl bych šanci jej stihnout dokončit během doby studia na střední škole. Na řadu tedy přišel můj druhý nápad - programovací jazyk. Na naší škole ještě nikdo programovací jazyk nevytvořil a já chtěl být první.

Napsat kompilátor, který by jazyk optimalizoval a kompiloval do procesorových instrukcí bych ale nestihl. Chtěl jsem, aby byl můj projekt multiplatformní a napsat kompilátor pro macOS, Windows i Linux bych nezvládl. Rozhodl jsem se tedy můj jazyk transkompilovat do jiného jazyka.

Jako výstupní jazyk jsem zvolil jazyk C. Vedla mě k tomu zejména znalost jazyka C, ale také to, že jazyk C je velmi malý, neobsahuje velké množství vestavěných funkcí, abstrakcí a syntaktického cukru. Díky tomu jsem mohl řadu funkcí a abstrakcí implementovat sám. Zároveň jsou programy napsané v jazyce C velmi rychlé.

Chtěl jsem programování v jazyce C zjednodušit, proto jsem se pokusil implementovat prvky z jiných programovacích jazyků.

Jedním z těchto prvků je přetěžování funkcí, které je vyřešené, podobně jako v jiných jazycích, přes name mangling, v mém jazyce však zjednodušený name mangling.

Další z novinek v mém jazyce jsou třídy. Třídy jsou implementovány jako struktury a podporují dědičnost. Můj jazyk však zatím nemá v-tables, tabulky s ukazateli na funkce, a polymorfismus tak zatím nefunguje.

Mezi další novinky patří např. zjednodušená syntax, automatické deklarace funkcí nebo dedukce datového typu. Z plánovaných funkcí se nabízí např. generické programování.

Pocházím z Kobeřic a jsem na to hrdý. Chci být nejlepší nejen kvůli sobě, ale také abych reprezentoval Kobeřice. Při výběru jména jsem se inspiroval jazykem Objective-C a svůj projekt jsem jako hrdý Kobeřan pojmenoval Kobeři-C.

2 PROGRAMOVACÍ JAZYKY A JEJICH PŘEKLADAČE

Počítače již od prvních generací fungují na principu dvou stavů, většinou označovaných čísly 0 a 1. Číselná soustava skládající se ze dvou čísel se nazývá binární, protože má dva stavy. Počítačové programy musí být napsané tak, aby jim procesor rozuměl, tedy v binární soustavě.

Po vzniku prvních počítačů se programy psaly přímo v binární soustavě. Tento způsob psaní programů byl ale velmi složitý. Vývoj programu trval dlouho, byl náročný a takto napsané programy se velmi špatně četly.

Proto vznikl jazyk nazývaný Assembly. Assembly se překládal do binárního kódu v poměru 1:1, každá instrukce v Assembly odpovídala jedné instrukci ve strojovém kódu. Výhodou Assembly bylo, že se k zápisu instrukcí používala slova připomínající lidskou řeč, proto se Assembly lépe četlo, jinak si ale Assembly ponechal všechny nevýhody psaní programů přímo ve strojovém kódu.

Za účelem dalšího zjednodušení programování a zrychlení vývoje vznikl jazyk Fortran, jenž poprvé vyšel v roce 1957. Od té doby vyšla spousta dalších jazyků, jako jsou např. C, C++, Lisp, Haskell, Julia a další.

Programovací jazyky měly za úkol zajistit práci s hardwarem, aby programátor mohl psát programy nezávisle na platformě. Tím, že programovací jazyky zajišťují práci s pamětí a registry procesoru, se zvyšuje produktivita programátora, který se může soustředit na logiku svého programu. Navíc tímto lze odstranit chyby, které by mohly vzniknout např. nepozorností programátora, který omylem přistoupí ke špatnému registru.

Programovací jazyky však připomínají jazyky lidské a procesory jim nerozumí. Aby počítač mohl program napsaný v určitém programovacím jazyce spustit, je třeba tento program nejprve přeložit.

2.1 Překladače jazyků

Překladače jsou programy, které jsou schopny přeložit programovací jazyky do binárních souborů spustitelných počítačem. Existuje několik typů překladačů: kompilátory, interpretery a transkompilátory.

Kompilátory překládají vstup v podobě kódu v programovacím jazyce do spustitelného souboru ve strojovém kódu procesoru, např. kompilátor G++ pro jazyk C++ nebo kompilátor GHC pro jazyk Haskell.

Interpretery překládají vstup a přímo jej spouští na virtuálním stroji. Program přitom vždy zůstává v podobě zdrojového kódu a při opětovném spuštění programu se kód musí znova přeložit. Příkladem interpretovaného jazyka je např. jazyk Python nebo Ruby.

Transkompilátory překládají zdrojový kód z jednoho jazyka do jiného programovacího jazyka. Výsledný výstupní soubor se poté předá překladači pro daný jazyk, který jej překládá dále. Příkladem je třeba jazyk TypeScript transkompilovaný do jazyka JavaScript nebo jazyk Kobeři-C.

2.1.1 Transkompilátory

Transkompilátory jsou překladače, které přijímají vstup v podobě zdrojového kódu v jednom jazyce a překládají jej do zdrojového kódu v jazyce jiném.

Transkompilátory můžou implementovat funkce, které výstupní jazyk postrádá. Tyto funkce se implementují buď staticky, anebo dynamicky.

Staticky implementované funkce nezpomalují chod programu. Např. přetěžování funkcí, dedukci datových typů nebo kontrolu datových typů lze provést již při kompilaci. Většinou se jedná o analýzu datových typů funkcí a proměnných, přetěžování funkcí, syntaktický cukr apod.

Dynamicky implementované funkce chod programu zpomalují. Proto se používají pouze když danou funkci nelze implementovat staticky. Příkladem těchto funkcí jsou dynamické typování ve staticky typovaných jazycích, polymorfismus a volání metod pomocí v-tabulek, garbage collector atd…

Transkompilátor je vhodné použít zejména v případě, kdy je výběr jazyků z nějakého důvodu omezený. Např. při tvorbě webových stránek, kde zatím nelze použít jiný jazyk než JavaScript, lze JavaScript nahradit jiným jazykem transkompilovaným do JavaScriptu (např. Typescript).

Výhodou této metody překládání programovacích jazyků je, že tvůrce transkompilátoru nemusí řešit překládání do strojového kódu, optimalizaci programu anebo tvorbu virtuálního stroje. Nevýhodou je limitace výstupním jazykem a nutnost obcházet omezení daného výstupního jazyka.

2.2 Princip fungování překladačů

Dnešní překladače jsou složité programy skládající se z více modulů, z nichž každý modul má určitou funkci.

2.2.1 Preprocesor

Preprocesor je spuštěn ještě před samotným překladem. Jak již název napovídá, preprocesor připravuje zdrojový kód na přeložení překladačem. Evaluuje preprocesorová makra, odstraňuje z kódu komentáře atd…

2.2.2 Tokenizer

Tokenizer čte zdrojový kód programu a provádí tokenizaci. Tokenizace je proces rozdělovaní textu do tzv. tokenů podle pravidel gramatiky jazyka.

2.2.3 Lexer

Lexer je modul kompilátoru, který provádí senzitivní analýzu tokenů a získává informace o tokenech v závislosti na kontextu a obsahu tokenů.

2.2.4 Parser a abstraktní syntaktický strom

Tokeny jsou následně rozparsovány a je vytvořen tzv. abstraktní syntaktický strom. Abstraktní synktaktický strom je datová struktura, která umožňuje ukládat kód jazyka v podobě, se kterou dokáže počítač lépe pracovat. Deklarace funkcí, tříd, instrukce atd. jsou uloženy abstraktně ve stromové struktuře.

2.2.5 Generátor kódu/interpreter

Poté, co je vytvořen syntaktický strom, je třeba jej přeložit do strojového kódu. Kompilátory a transkompilátory obsahují generátory kódu, které dokáží strom přeložit do strojového kódu, popř. jiného jazyka. Interpretery obsahují virtuální stroj, na kterém lze daný program přímo spustit.

3 ZPŮSOBY ŘEŠENÍ, VYUŽITÉ POSTUPY A TECHNOLOGIE

K vytvoření transkompilátoru jazyka Kobeři-C jsem použil jazyk C++11. Samotný kód nemá žádné externí dependence a využívá pouze standardní knihovny jazyka C++, včetně STL.

3.1 C++11

Kompilátor svého jazyka jsem napsal v jazyce C++, protože jej znám ze všech jazyků nejlépe. Není to však můj nejoblíbenější jazyk a často jsem si přál, abych mohl používat nějaký jiný jazyk, např. Haskell nebo C#.

3.2 Nevyužité technologie

Zvažoval jsem i použití externích dependencí a existujících technologií, jako jsou generátory parserů nebo LLVM. Abych tyto technologie mohl použít, musel bych se je nejprve naučit používat. To by přidalo zbytečnou časovou komplexitu k již tak dost časově náročné tvorbě kompilátoru a já bych projekt nestihl dokončit.

3.2.1 Generátory parserů

Generátory parserů jsou programy, které umožňují generování parserů. Vstupem těchto generátorů je gramatika jazyka zapsaná v předem dané notaci, výstupem je zdrojový kód parseru v určitém jazyce. Generátor jsem nakonec nevyužil, protože jsem si sám chtěl vyzkoušet psaní parseru a neměl jsem čas učit se gramatickou notaci generátoru. Kdybych ale generátor parseru využil, zvolil bych generátor Yacc nebo GNU Bison.

3.2.2 LLVM

Přemýšlel jsem také nad použitím LLVM ke kompilaci programů napsaných v jazyce Kobeři-C do nativních spustitelných souborů. Abych však mohl využít LLVM, musel bych kód zkompilovat do jazyka LLVM, který by následně byl zkompilován platformou LLVM. Jazyk LLVM mi ale přišel složitý, strávil bych spoustu času učením se jazyka LLVM, proto jsem nakonec LLVM nevyužil. Kompilátory spousty jazyků, jako například kompilátory jazyků Common Lisp, Kotlin, C++ nebo Swift, využívají technologii LLVM. Výhodou LLVM je vysoký výkon programů zkompilovaných pomocí LLVM, jelikož LLVM se zaměřuje především na optimalizaci kompilovaných programů.

4 KOMPILÁTOR KOBEŘI-C

Kompilátor Kobeři-C se skládá ze čtyř hlavních částí – Tokenizeru, Parseru, Analyzeru a Translatoru. Dále také obsahuje Name Mangler, systém importů a dva různé abstraktní syntaktické stromy.

4.1 ImportSystem

Systém importů, v mém kompilátoru reprezentovaný třídou ImportSystem, zajišťuje import knihoven jazyka Kobeři-C, knihoven jazyka C a externích datových typů. ImportSystem rekurzivně prochází všechny soubory, hledá v nich import direktivy a vytváří seznamy všech importovaných souborů a datových typů.

4.2 Tokenizer

Třída Tokenizer zajišťuje nejen tokenizaci, ale i lexování všech importovaných souborů obsahujících kód napsaný v jazyce Kobeři-C. Třída Tokenizer tímto spojuje tokenizer s lexerem. Výstupem Tokenizeru je pole tokenů, které obsahuje tokeny ze všech importovaných souborů zároveň.

4.3 Parser

Tokenizovaný kód je následně rozparsován třídou Parser. Parser prochází pole tokenů a rekurzivně z něj vytváří abstraktní syntaktický strom.

4.4 AbstractSyntaxTree

Abstraktní syntaktický strom je reprezentován dvěma třídami – třídou AbstractSyntaxTree a jejím rozšířením TraversableAbstractSyntaxTree. Třída AbstractSyntaxTree obsahuje metody sloužící k tvoření syntaktického stromu. Třída TraversableAbstractSyntaxTree dědí z třídy AbstractSyntaxTree a rozšiřuje ji o možnost přístupu k uzlům stromu, datovým typům atd.

4.5 Analyzer

Po vytvoření syntaktického stromu je třeba tento strom projít, analyzovat a nalézt případné chyby ve zdrojovém kódu. Toto zajišťuje třída Analyzer, která syntaktický strom rekurzivně prochází, kontroluje datové typy, kontroluje, zda volané funkce existují a zajišťuje zavolání správného přetížení funkce, zajišťuje volání destruktorů atd. Výstupem Analyzeru je další strom, AnalyzedAbstractSyntaxTree.

4.6 AnalyzedAbstractSyntaxTree

Analyzovaný abstraktní syntaktický strom je kolekce obsahující validní analyzovaný program připravený k přímému překladu do jazyka C. Uzly tohoto stromu se dokážou samy rekurzivně přeložit.

4.7 Translator

Třída Translator zajišťuje správný průběh překladu a zápis výstupu do souboru. Nejprve zapíše do souboru *#import* makra preprocesoru jazyka C, deklarace tříd, funkcí a globálních proměnných a nakonec definice funkcí.

4.8 NameMangler

Třída NameMangler je statická třída, která obsahuje pouze dvě členské funkce. První členská funkce zajišťuje name mangling podle datových typů parametrů funkce. Před název funkce je připojena předpona společná pro všechny funkce napsané v jazyce Kobeři-C. Za název funkce jsou připojeny názvy datových typů parametrů oddělených podtržítky. Druhá funkce zajišťuje name mangling podle názvu třídy a používá se u členských funkcí, před jejíž název je připojen název třídy.

5 PŘEHLED JAZYKA KOBEŘI-C

5.1 Základní filozofie jazyka Kobeři-C

Jazyk Kobeři-C je rozpoznatelný díky své jednoduché syntaxi velmi podobné jazyku LISP. Syntax jazyka Kobeři-C je velmi jednoduchá k naučení, protože je unifikovaná a pro všechny struktury jazyka Kobeři-C se výrazně podobá. Vše se zapisuje do tzv. s-výrazů ohraničených kulatými závorkami.

Jazyk Kobeři-C přináší do jazyka C kromě zjednodušené syntax a syntaktického cukru také nové abstrakce, jako jsou třídy nebo přetěžování funkcí.

5.2 Funkce

Funkce jsou v jazyce Kobeři-C automaticky předem deklarovány. To znamená, že např. na řádku 50 lze zavolat funkci deklarovanou až na řádku 60. Funkce jsou také přetěžovány podle datových typů parametrů. Funkce v jazyce Kobeři-C není možné definovat uvnitř jiné funkce, nelze je přetěžovat pomocí návratového typu a nepodporují pojmenované nebo výchozí parametry.

Definice funkcí v jazyce Kobeři-C začínají návratovým typem následovaným názvem funkce. Po názvu následují parametry funkce zapsané jako s-výraz a poté následuje tělo funkce. Tělo funkce se zapisuje jako libovolný počet s-výrazů.

Funkce jsou volány zapsáním do s-výrazu. První hodnota v s-výrazu je název volané funkce, který je následován parametry. Operátory jsou volány stejným způsobem.

Všechny programy v jazyce Kobeři-C musí obsahovat funkci *(int main ())*, která je zavolána při spuštění programu.

```
;;; Definice funkce
(int pow (int x int exp)
    (if (not exp)
        (return 1))
    (return (* x (pow x (- exp 1)))))

;; Volání funkcí
(factorial x)
(print "Factorial of " 5 " is " (factorial 5))
(and value1 value2)
```

5.3 Datové typy

Jazyk Kobeři-C obsahuje 7 základních datových typů: *void*, *int* (64bitové celé číslo), *uint* (64bitové nezáporné celé číslo), *char* (8bitový ASCII znak/celé číslo), *uchar* (8bitový ASCII znak/nezáporné celé číslo) a *num* (64bitové reálné číslo).

Je také možné definovat vlastní uživatelské datové typy (třídy), nebo importovat datové typy z jazyka C.

Ke každému datovému typu existují také tzv. ukazatele. Ukazatele fungují podobně jako v jazyce C a deklarují se pomocí znaku ‚*‘, který musí následovat hned za datovým typem (např. *int**). Ukazatele jsou automaticky dereferencovány, pokud není přístup na adresu explicitně vyjádřen operátorem *&* nebo pokud ukazatel nenastavujeme na hodnotu jiného ukazatele.

Hodnoty lze přetypovat pomocí operátoru *(cast)*. Operátor *(cast)* přijímá dva parametry. Prvním parametrem je hodnota, kterou chceme přetypovat, druhým parametrem je datový typ, na který chceme danou hodnotu přetypovat. Pro přetypování hodnot existují daná pravidla. Ukazatele lze přetypovat na jakýkoliv jiný ukazatel. Primitivní numerické typy lze přetypovat na jakýkoliv jiný primitivní numerický typ. Instance tříd lze přetypovat pouze na třídy, ze kterých daná třída dědí nebo na třídy dědící z dané třídy.

```
;; Přístup na adresu hodnoty
(& value)

;; Přetypování hodnoty
(cast 6 num)
(cast 62 char)
(cast child_instance ParentClass)
```

5.4 Proměnné

Jazyk Kobeři-C je staticky typovaný a datový typ proměnné tedy nelze po deklaraci změnit. Proměnné se do s-výrazu zapisují podobně jako funkce, nejprve se zapíše datový typ, poté název a nakonec nepovinná hodnota pro inicializaci proměnné. Atributy tříd a globální proměnné nelze takto inicializovat.

Proměnné lze deklarovat manuálním specifikováním datového typu, ale také lze použít tzv. dedukci datových typů (type inference). Pomocí klíčového slova *var* lze deklarovat proměnnou, jejíž datový typ je dedukován při kompilaci podle hodnoty přiřazené této proměnné. Použití klíčového slova *var* vyžaduje, aby proměnné byla přiřazena hodnota, podle které lze datový typ dedukovat.

```
;; Deklarace proměnné
(int x)
(int x2 10)
(int* x_ptr (& x))

;; Dedukce datového typu
(var integer (get_int)) ; int
(var car (new Car)) ; car*
```

5.5 Třídy

Výhodou jazyka Kobeři-C oproti jazyku C je přítomnost tříd. Třídy se deklarují pomocí klíčového slova *class* a můžou obsahovat atributy i metody. V metodách lze na instanci dané třídy odkazovat pomocí ukazatele *self*, který je metodě předán implicitně. Třídy v jazyce Kobeři-C podporují dědičnost a částečně polymorfismus, nepodporují ale privátní a statické členy, a prozatím ani dynamické linkování metod.

Instance tříd lze vytvořit deklarováním proměnné, nebo pomocí operátoru *(new)*, který alokuje paměť na haldě a vrací ukazatel na nově vytvořenou instanci. Instance lze smazat pomocí operátoru *(delete)*.

K atributům a metodám lze přistupovat pomocí operátoru *[]*, do kterého se zapíše, k jakému atributu nebo metodě přistupujeme. Operátor *[]* přijímá libovolný počet parametrů. Např. *[(get_object) nested_object attribute]*.

Členské funkce (metody) lze volat stejně jako ostatní funkce. První parametr s-výrazu specifikuje metodu pomocí operátoru *[]* a další parametry s-výrazu jsou předány jako parametry dané metodě. Např. *([object method] 1 2 3)*.

Ke třídám lze vytvořit také destruktory, metody, které jsou zavolány automaticky, když je opuštěn s-výraz, ve kterém byly instance třídy deklarovány, když je zavolán *return*, nebo když je instance smazána pomocí *(delete)*, ale lze je také zavolat manuálně. Destruktory jsou deklarovány jako metody s názvem *destruct*, které nepřijímají žádný parametr.

```
;;; Definice třídy
(class Shape ()
    ;; Definice metody
    (int area ()
        (return 0)))

;;; Definice třídy dědící od jiné třídy
(class Square (Shape)
    (int side)
    (int area ()
        (return (* [self side] [self side])))
    ;; Definice destruktoru
    (void destruct ()
        (print "Destructing square")))

;; Vytvoření nového objektu typu String na
;; haldě
(String* str (new String))
;; Smazání objektu str
(delete str)

(Square s)
;; Přístup k atributu objektu
[s side]
;; Volání metody objektu s
([s area])
```

5.6 Řídící struktury

Základními řídícími strukturami v jazyce Kobeři-C jsou struktury *while*, *dowhile* a *if*/*elif*/*else*. Zapisují se podobně jako vše ostatní v jazyce Kobeři-C - nejprve se napíše název struktury, poté podmínka a nakonec se do výrazu zapíše libovolný počet příkazů.

Větve *elif* a *else* jsou nepovinné a nemusí se objevit za každým výrazem *if*.

Výraz *dowhile* se zapisuje stejně jako výraz *while*, ale podmínka je zde kontrolována až po dokončení jedné iterace cyklu, funguje tedy stejně jako výraz *do {} while ()*; v jazyce C.

Příkaz *for* zatím v jazyce Kobeři-C neexistuje.

```
;; Příkaz if/elif/else
(if x
    (print x " is true. \n"))

(if (> x y)
    (print x " is greater than " y "\n"))
(elif (equals x y)
    (print x " equals " y "\n"))
(else
    (print x " is less than " y "\n"))

;; Příkaz while, dowhile
(while (< i 5)
    (inc i))

(dowhile (not_eq x 0)
    (dec x)
    (print x "\n"))
```

5.7 Syntaktický cukr

Programovací jazyk Kobeři-C s sebou přináší také syntaktický cukr, který zpřehledňuje a zjednodušuje zdrojový kód. Mezi syntaktický cukr jazyka Kobeři-C se řadí např. variadičnost funkcí *equals*, <, >, +, -, *, / atp. Tyto funkce jsou variadické, přijímají tedy libovolný počet parametrů a evaluují se postupně. Např. *(equals a b c d)* se přeloží na *(a == b && b == c && c == d)* nebo *(+ a b c d)* se přeloží na *(a + b + c + d)*. Funkci *(-)* lze použít k aritmetické negaci hodnot, pokud funkci předáme pouze jeden parametr.

Výpis do stdout je oproti jazyku C také jednodušší. Místo funkcí jako *puts()*, *putchar()* a formátovaného výstupu pomocí *printf()* zde existuje pouze jedna funkce - *(print)*. *(Print)* je variadická funkce, které lze předat parametry typu *int*, *uint*, *char*, *uchar*, *num* a *char**. Všechny parametry jsou poté vypsány v pořadí, v jakém byly funkci předány.

```
;;; Příklady použití variadických funkcí

(and value1 value2 (get_value))
(+ 1 2 3 4 5 6.2)
(mod 123 23 12 3 2)
(print "Hodnota proměnné x je: " x " x! je "
       (factorial x) "\n")
```

5.8 Komentáře

Komentáře v jazyce Kobeři-C začínají znaky „;“ nebo „#“. Je však doporučováno, aby komentáře začínaly znakem „;“, protože řádky začínající znakem „#“ mají v jazyce Kobeři-C speciální význam.

Pro psaní komentářů se doporučuje dodržovat konvence zavedené jazykem Lisp, kde čtyři středníky značí komentář popisující program, tři středníky značí komentář začínající na začátku řádku, popisující např. funkci nebo třídu, dva středníky se používají pro komentáře odsazené společně s kódem a jeden středník pro komentáře na konci řádku.

```
;;;; Příklad použití komentářů v jazyce Kobeři-C

;;; Funkce multiply - vrací násobek dvou čísel
(int multiply (int x int y)

    ;; Výpočet výsledku
    (int result (* x y))
    (return result)) ; Vracení výsledku
```

5.9 Import

Moduly jazyka Kobeři-C lze importovat pomocí výrazu *#import*, kterému lze předat libovolný počet parametrů. Parametry výrazu import se zapisují do uvozovek a za názvy souborů se nedopisuje přípona.

Lze také importovat knihovny jazyka C, které se importují pomocí výrazu *#extern*. Výrazu *#extern* lze předat libovolný počet parametrů. Standardní knihovny jazyka C se uvozují pomocí znaků <>, ostatní knihovny jsou zapisovány v uvozovkách.

Výraz *#extern* dokáže importovat také externí datové typy, které lze poté použít v jazyce Kobeři-C. Takto lze externí datové typy použít např. k deklaraci proměnných, nelze s nimi ale provádět žádné operace specifické pro daný datový typ, jako např. přístup k atributům. Externí datové typy se nijak neuvozují.

```
;; Import modulů jazyka Kobeři-C
#import "String" "lib/Math"

;; Import hlavičkových souborů jazyka C
#extern <iso646.h> <thread.h> "lib.h"

;; Import externího datového typu
#extern FILE

;; Parametry lze libovolně řetězit
#extern <thread.h> "lib.h" FILE <iso646.h>
```

5.10 Vkládané C

Kobeři-C podporuje tzv. vkládané C. Do zdrojového kódu jazyka Kobeři-C tak lze pomocí operátoru *(_c)* vložit kód napsaný v jazyce C. Díky tomu lze např. zavolat funkce z knihoven jazyka C, které nepodléhají přetěžování funkcí jazyka Kobeři-C. Operátor *(_c)* přijímá libovolný počet parametrů. Operátoru *(_c)* lze předat pouze literály řetězců. Vkládané C lze použít pouze uvnitř těl funkcí.

```
;;;; Příklad použití vkládaného C

;;; Funkce malloc, která podléhá name-manglingu
;;; Lze ji zavolat jako funkci jazyka Kobeři-C
;;; Interně ale volá funkci malloc() z jazyka C

(void* malloc (int size)
    (void* ptr)
    ;; Alokace paměti a přiřazení adresy
    ;; do proměnné ptr
    (_c "ptr = malloc(size);")
    (return (& ptr)))
```

6 ZÁVĚR

Cílem projektu bylo vytvořit vlastní funkční programovací jazyk Kobeři-C. Určeným cílem bylo vytvoření kompilátoru o čtyřech průchodech zdrojovým kódem, který přeloží kód v jazyce Kobeři-C do jazyka C. Jazyk Kobeři-C měl podporovat přetěžování funkcí, alespoň částečně objektově orientované programování, import modulů, ukazatele, měl se oproti jazyku C lišit syntaxí a měl obsahovat syntaktický cukr.

Všechny mnou určené cíle byly splněny. Programovací jazyk Kobeři-C je nyní použitelný k programování skutečných programů a aplikací.

Do budoucna bych ještě rád ve vývoji jazyka pokračoval. Přidal bych další funkce, které jazyku chybí, jako výčty, konstanty nebo dynamické volání metod.

7 POUŽITÁ LITERATURA

[1] KYLE, James - The Super Tiny Compiler (GitHub) [online].
[cit. 2017-12-15]
Dostupné z: https://github.com/thejameskyle/the-super-tiny-compiler

[2] LLVM PROJECT - LLVM Tutorial: Kaleidoscope (LLVM Project) [online].
[cit. 2017-12-15]
Dostupné z: http://llvm.org/docs/tutorial/

[3] THE CLANG TEAM - "Clang" CFE Internals Manual (LLVM Project) [online].
[cit. 2017-12-15]
Dostupné z: http://clang.llvm.org/docs/InternalsManual.html

[4] 9000 - How to write a very basic compiler (Stack Exchange) [online].
[cit. 2017-12-15]
Dostupné z: https://softwareengineering.stackexchange.com/a/165558

8 Příloha 1: Ukázka jazyka Kobeři-C

```
;;;; Ukázka jazyka Kobeři-C
;;;; Program, který vypíše čísla 1 až 20 a jejich
;;;; druhou mocninu

(int pow (int x int exp)
    (if (not exp)
        (return 1))
    (return (* x (pow x (dec exp)))))

(int main ()
    (var i 1)
    (while (<= i 20)
        (print i " - " (pow i 2) "\n")
        (inc i)))

/* Výstup v jazyce C                                  */
/* Zobrazeny jsou pouze relevantní části výstupu */

typedef intmax_t int_type;

int_type _koberic_pow_int_int(int_type x, int_type exp);
int_type _koberic_main();

int_type _koberic_pow_int_int(int_type x, int_type exp)
{
    if (!( exp ))
    {

        {
            return( 1ll );
        }

    }


    {
        return( (x * _koberic_pow_int_int(x, --( exp ))) );
    }

}
```

```
int_type _koberic_main()
{
    int_type i = 1ll;
    while (i <= 201ll)
    {
        printf("%lld", i);
        fputs(" - ", stdout);
        printf("%lld", _koberic_pow_int_int(i, 2ll));
        fputs("\n", stdout);
        ++( i );
    }

}



/* C Main Function */

int main(int argc, const char * argv[]) {
    return _koberic_main();
}
```