Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Objektivní analýza výkonu překladačů vybraných programovacích jazyků

Plzeň 2016 Lukáš Haringer

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 23. června 2016

Lukáš Haringer

Abstract

This bachelor thesis deals with an unbiased analysis of compiler performance for selected programming languages. The aim of this work is to choose appropriate algorithms for testing the performance of compilers and codes that produce. These algorithms then implement in the selected programming languages and compile in several different compilers. The work also describes the measurement procedure and discusses the results.

Abstrakt

Tato bakalářská práce se zabývá objektivní analýzou výkonu překladačů vybraných programovacích jazyků. Cílem této práce je vybrat vhodné algoritmy tak, aby otestovaly výkon překladačů a kódů, které vyprodukují. Tyto algoritmy pak co nejpodobněji implementovat ve vybraných programovacích jazycích a přeložit několika různými překladači. Následně detailně porovnat a popsat dosažené výkony.

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Ing. Kamilu Ekšteinovi, Ph.D. za jeho vedení, trpělivost a cenné rady v průběhu vypracovávání celé této práce.

Obsah

1 Úvod

Počítače nám v dnešní době pomáhají na každém kroku. Ať už za nás řeší složité matematické výpočty, uchovávají obrovské množství dat, které by lidský mozek nebyl schopen pojmout nebo nám pouze umožňují komunikaci s přáteli. Nic z toho by však počítač nemohl dokázat bez svého softwaru. Většina softwaru je dnes psaná ve vyšších programovacích jazycích. Počítač však není schopen program zapsaný v takovém jazyce vykonat, protože CPU počítače je schopen pouze zpracovat instrukce strojového kódu. Takový zápis je ale pro člověka nepohodlný, špatně čitelný a závislý na daném typu počítače. Z tohoto důvodu se dnes používají vyšší programovací jazyky, které následně překladače přeloží do podoby spustitelného kódu na dané platformě. Dnes existuje řada překladačů pro různé programovací jazyky a každý generuje jinak výkonný strojový kód.

Cílem této práce je vybrat vhodné algoritmy tak, aby otestovaly výkon překladačů a kódů, které vyprodukují. Tyto algoritmy pak co nejpodobněji implementovat ve vybraných programovacích jazycích a přeložit několika různými překladači. Následně detailně porovnat a popsat dosažené výkony.

Po přečtení této práce získáte představu o principech fungování překladačů, způsobech profillingu výsledných kódů, rozdílů mezi základními typy překladačů a o výhodách a nevýhodách používání konkrétních překladačů.

2 Programovací jazyky

Programovací jazyky jsou způsobem, kterým se mezi sebou mohou domluvit programátor a počítač, tedy člověk a stroj. Běžné jazyky, kterými mezi sebou komunikují lidé, jsou plné neurčitostí a nepřesností, a tedy pro programování zcela nevhodné. Na druhou stranu strojový kód, který využívají počítače, je zcela určitý, ale pro člověka špatně čitelný a náročný na zapamatování. Vývoj programu v takovém jazyce trvá dlouhou dobu, a proto se v dnešní době využívá už jen výjimečně. Z tohoto důvodu vznikly nové programovací jazyky, které jsou pro člověka lépe srozumitelné a vývoj programů v těchto jazycích trvá mnohem kratší dobu. Programy zapsané v těchto jazycích však musejí být před zpracováním procesorem počítače převedeny do strojového kódu, který má binární tvar.[5]

2.1 Formální jazyk

Definice: Libovolná neprázdná množina V je abecedou. Její prvky nazýváme znaky nebo symboly.[1]

Definice: Slovem nad abecedou V myslíme konečnou posloupnost znaků z V , značíme $w = a_1 + a_2 + \cdots + a_n$.[1]

Definice: Jazykem L nad abecedou V rozumíme libovolnou (ne nutně konečnou!) množinu slov, tedy $L \subseteq V^*$[1]

2.2 Dělení programovacích jazyků dle abstrakce

Programovací jazyky se dají rozdělit podle mnoha kritérií. Nejčastěji se však dělí podle míry abstrakce. Nízká míra abstrakce znamená malý nebo žádný rozdíl mezi daným programovacím jazykem a strojovým kódem.

2.2.1 Nižší programovací jazyky

Nižší programovací jazyky jsou jazyky primitivní, jejichž instrukce víceméně přesně odpovídají instrukcím daného procesoru. Mají velmi nízkou míru abstrakce. Programy napsané v těchto jazycích jsou rychlé, mívají nízké nároky na paměť počítače a programátor má přístup k funkcím procesoru, které by byly ve vyšším programovacím jazyce nedostupné. Hlavními nevýhodami těchto programovacích jazyků jsou nepřenositelnost mezi různými typy procesorů, horší osvojitelnost a i jednoduché programy zapsané v těchto jazycích bývají velmi dlouhé. Mezi tyto jazyky patří například Assembler.

2.2.2 Vyšší programovací jazyky

Vyšší programovací jazyky, též problémově orientované jazyky, mají velkou míru abstrakce. To znamená, že nejsou tak úzce spjaté s hardwarem počítače, a proto jsou snáze přenositelné mezi jednotlivými typy procesorů. Programy zapsané v těchto jazycích jsou pro člověka mnohem lépe čitelné a vývoj programu probíhá mnohem rychleji než u nižších programovacích jazyků. Na druhou stranu tyto programy jsou většinou pomalejší a mají větší paměťovou náročnost.

S příchodem vyšších programovacích jazyků vznikla potřeba konstrukce speciálních programů pro transformaci mezi daným vyšším programovacím jazykem a cílovým jazykem daného stroje. Zároveň vznikly dvě možnosti, jak přistupovat k překladu kódu: přístup kompilační a přístup interpretační. Podle tohoto rozdělení se dodnes dělí překladače na kompilátory a interpretry. [4]

2.3 Dělení programovacích jazyků dle paradigmat

Programovací paradigma je základní programovací styl. Vyšší programovací jazyky lze rozdělit do skupin, podle toho jaké programové paradigma je v daném jazyce nejčastěji používané. Programovací jazyk může podporovat více než jedno paradigma. Jako příklad lze uvést jazyk C++, jehož programy mohou být psány čistě procedurálně, čistě objektově nebo jako kombinace

obou. [6]

2.3.1 Naivní paradigma

Naivní paradigma nejčastěji používají počítačoví začátečníci. Charakteristickými vlastnostmi jsou minimální strukturovanost, neumožňují modularitu a mají minimální datovou abstrakci. [4]
Použití: Toto paradigma není vhodné používat.

2.3.2 Funkcionální paradigma

Jedná se o druhé nejstarší paradigma. Výpočet probíhá postupným aplikováním funkcí jedné na druhou. Hlavním znakem je nepoužívání přiřazovacích příkazů a velký důraz na rekurzi. [4]
Jazyky: Lisp

2.3.3 Logické paradigma

V logickém paradigmatu je program množina takzvaných klauzulí. Jedná se o množinu faktů a pravidel, ze kterých se poté systém snaží za pomoci zpětného zřetězení dokázat nebo vyvrátit tvrzení předložené programátorem. [4]
Jazyky: Prolog

2.3.4 Procedurální paradigma

Jedná se o nejstarší paradigma. Průběh výpočtu je dán sekvencí po sobě jdoucích příkazů, které určují, jak danou úlohu řešit. Program je sadou proměnných, které následkem vyhodnocování podmínek mění svůj stav. [4]
Jazyky: C, Pascal, Fortran
Použití: Toto paradigma je velmi univerzální a rozšířené zejména v komerční sféře.

2.3.5 Objektové paradigma

Objektové paradigma je založeno na modelování předmětů reálného světa do podoby takzvaných objektů. Každý objekt má své atributy a metody. Atributy uchovávají informace o daném objektu a metody jsou schopny na požádání tyto atributy změnit. Výpočet probíhá posíláním zpráv mezi jednotlivými objekty. [4]
Jazyky: Ruby, C#, Java, Python, Smalltalk
Použití: Původní použití se soustředilo na oblast umělé inteligence, v moderních jazycích je použití již velmi univerzální.

2.3.6 Paralelní paradigma

Princip paralelní paradigmatu spočívá ve snaze dekomponovat algoritmus na součásti, které mohou být zpracovány současně odlišnými procesory. V prostředí paralelního programování je třeba dbát na synchronizování výpočtů a ošetření kritických sekcí, aby nedošlo ke špatnému zpracování dat. [4]
Jazyky: C#, Java

3 Překladače

Každý člověk, který někdy napsal program v nějakém vyšším programovacím jazyku, se setkal s překladačem. Ne každý však ví, jak funguje. Mnoho programátorů bere překladač pouze jako černou skříňku, která transformuje jejich kód do podoby kódu spustitelné na dané platformě. Pojd'me se tedy podívat, jak proces této transformace funguje.

Překladač je program, který ke zdrojovému programu ve vyšším programovacím jazyce vytvoří cílový program ve spustitelném kódu dané platformy se stejným významem.

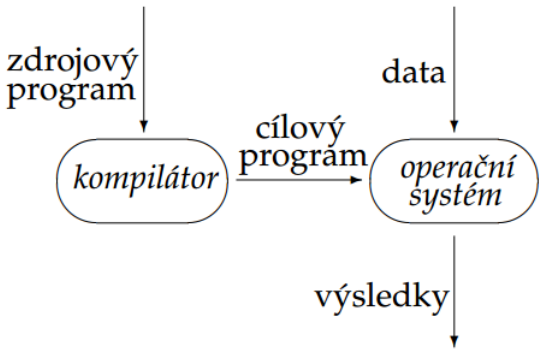
3.1 Rozdělení překladačů podle typu cílového programu

3.1.1 Kompilátor

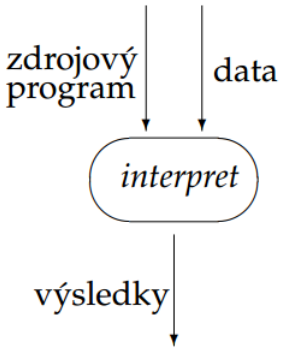
Kompilátor je překladač, který z programu ve vyšším programovacím jazyce (C, C++, Pascal...) vytvoří strojový jazyk spustitelný na dané platformě. U kompilátorů samotný překlad probíhá jen jednou a nesouvisí se samotným během programu, což umožňuje provádět i časově náročné optimalizace a kontroly. [3]

3.1.2 Interpret

Interpret, někdy také interpreter, na rozdíl od kompilátoru negeneruje žádný spustitelný program. Pouze interpretuje příkazy zdrojového jazyka a provádí příslušné akce. Interpretace se provádí při každém spuštění programu. Řadíme sem například skriptovací jazyky Python, PHP a Perl. [3]



Obrázek 3.1: Kompilační překladač [3], upraveno autorem.



Obrázek 3.2: Interpretační překladač [3], upraveno autorem.

3.1.3 Hybridní překladače

Jak vyplývá z názvu, hybridní překladače jsou směsí interpretačního a kompilačního přístupu. Zdrojový kód vyššího programovacího jazyka nejdříve kompilátor hybridního překladače převede na mezikód, který je nezávislý na operačním systému. Tento mezikód je následně interpretován interpretační částí překladače, která je nainstalována na počítači, na kterém chceme mezikód spustit. Typickými zástupci jsou jazyky Java a C#. [3]

3.1.4 Výhody a nevýhody jednotlivých druhů překladačů

Každý z těchto druhů překladačů má své silné a slabé stránky. Zatímco u kompilátorů se výsledný přeložený program provádí relativně rychle u interpretů je běh pomalejší kvůli potřebě překladu při každém spuštění, a proto nevhodný pro mnoho vyšších programovacích jazyků. Další nevýhodou interpretovaných jazyků je potřeba nainstalování daného interpretu na počítač, na kterém chceme daný kód spouštět. Programy napsané v interpretovaných jazycích jsou také více paměťově náročné, protože při běhu programu musí být v operační paměti uložen jak program tak i interpretr. Při velikostech pamětí RAM, jaké mají dnešní počítače nám však toto nevadí tak jako dříve a některé nejpoužívanější interprety jsou dokonce základní součástí některých linuxových distribucí. Výhodou interpretových překladačů je, že ten samý kód je možné spustit na mnoha platformách. Pouze je potřeba nainstalovat překladač pro danou platformu. Navíc program interpretru má většinou pouze textovou podobu a zabírá mnohem méně místa na uživatelově disku než kompilované programy. [3]

Vlastnost	**Kompilátor**	**Interpret**
Rychlost běhu cílového programu	lepší	
Rychlost spuštění cílového programu	lepší	
Rychlost překladu		lepší
Spotřeba paměti - operační	lepší	
Spotřeba paměti - program na paměťovém médiu		lepší
Přenositelnost kódu mezi platformami		lepší
Možnost optimalizace	lepší	
Nezávislost na překladači	lepší	

Tabulka 3.1: Srovnání vlastností kompilačního a interpretačního překladače. [3]

3.2 Části překladače

Podle závislosti na typu cílového kódu můžeme překladač rozdělit na dvě základní části. Přední část, která obsahuje lexikální, syntaktickou a séman-

tickou analýzu. Přední část je do značné míry nezávislá na cílové platformě. Zadní část se stará o optimalizaci kódu a generování cílového programu. Tato část je již závislá na cílové platformě. [3]

Díky tomuto rozdělení při vytváření překladačů téhož jazyka pro různé platformy nemusíme vytvářet pokaždé znovu přední část překladače, která je stejná pro všechny platformy. Překladače se budou lišit pouze v zadní části, která generuje kód pro danou platformu. [3]

Překladač lze dále rozdělit na části, z nichž každá má při překladu jiný úkol. Činnost těchto částí nazýváme fáze překladu. Těmito částmi jsou: [3]

1. lexikální analyzátor
2. syntaktický analyzátor
3. sémantický analyzátor
4. optimalizátor kódu
5. generátor cílového kódu nebo interpretace.

3.2.1 Lexikální analyzátor

Lexikální analyzátor je první částí překladače, která přijde do styku s naším kódem. Jeho účelem je transformovat zdrojový kód do podoby vhodné pro zpracování dalšími částmi překladače. Lexikální analyzátor vyřazuje z kódu znaky, které pro nás nemají smysl. Těmito znaky jsou například nevýznamové mezery nebo komentáře. Dále syntaktický analyzátor rozdělí kód na takzvané atomy (logicky nejmenší části s vlastním významem) a každému přiřadí identifikátor. [3]

3.2.2 Syntaktický analyzátor

Syntaktický analyzátor je patrně nejdůležitější částí překladače. Úkolem syntaktického analyzátoru je vytvořit strukturu daného programu, která je obvykle reprezentována derivačním stromem v nějaké vhodné reprezentaci. Syntaktický analyzátor zjišťuje, jak jednotlivé atomy vygenerované lexikálním analyzátorem patří k sobě a tvoří z nich příkazy, definice proměnných

a další struktury. [3] Během této fáze překladu je také kontrolována syntaktická správnost programu.

3.2.3 Sémantický analyzátor

Sémantický analyzátor je poslední částí přední části překladače. Přiřazuje význam každé skupině symbolů získané syntaktickou analýzou. To znamená, že kontroluje práci programu s datovými typy. Například při zpracování deklarace proměnné zkontroluje, zda není již deklarována a uloží do patřičného seznamu potřebné informace (název, typ, hodnotu ...) a může také přiřadit paměť. [3] Výstupem sémantického analyzátoru je mezikód, který je vhodný pro optimalizaci, nebo se dá u interpretačních překladačů použít pro interpretaci.

3.2.4 Optimalizátor kódu

Optimalizátor kódu se snaží vylepšit mezikód poskytnutý sémantickým analyzátorem tak, aby z něj mohl vzniknou lepší strojový kód. Lepším většinou myslíme rychlejší, ale v některých případech se může snažit například i o zmenšení místa které bude výsledný program zabírat na disku. Optimalizátor kódu zajišťuje, aby v programu bylo použito co nejméně pomocných proměnných, aby se nevyhodnocoval zbytečně v cyklu tentýž příkaz a podobně. Optimalizace se typicky provádí u kompilátorů, interpreti optimalizaci většinou přeskakují. [3]

3.2.5 Generátor cílového kódu nebo interpretace

Jedná se o poslední část překladače. Z optimalizovaného mezikódu, vytvořeného předešlými fázemi překladu, vytvoří výsledný kód buď v podobě vlastního jazyka sestavujícího programu (interpret) nebo strojovém jazyce (kompilátor).

4 Vybrané jazyky

Tato kapitola bude věnována vybraným jazykům, jejichž překladače budou testovány v této práci. Těmito vybranými jazyky jsou C, C++, C#, Object Pascal, Java a Scalla. Každému jazyku bude věnován krátký popis, ukázka syntaxe programu a poté bude následovat výčet překladačů daného jazyka, které budou testovány.

4.1 Ansi/ISO C

C je kompilovaný, nízkoúrovňový programovací jazyk vyvinutý Kenem Thompsonem a Dennisem Rithchiem mezi lety 1969 a 1973 ve společnosti AT&T Bell Labs. C je v současné době jedním z nejpoužívanějších jazyků. Nejčastěji je využíván pro psaní operačních systémů, překladačů a systémových ovladačů. Jazyk C pracuje s pamětí přes datový typ ukazatel (anglicky pointer). Ukazatelé drží odkaz na paměťový prostor dané proměnné. S ukazateli je možné provádět aritmetické operace a je jen na zodpovědnosti programátora, aby neodkazovali na nealokovanou paměť. Jazyk C nemá garbage collector a programátor je tedy odpovědný za dealokaci alokované paměti. Jazyk C byl v roce 1989 standardizován organizacemi American National Standards Institute (ANSI) a International Organization for Standardization (OSI)

Ukázka syntaxe jazyka:

```
/* Hello World program */

#include<stdio.h>

main()
{
    printf("Hello World");
return 0;
}
```

Testované překladače:

- GCC
- Microsoft Visual C++
- Intel C++

4.2 C++

C++ je kompilovaný programovací jazyk, který byl vyvinut v roce 1983 Bjarnem Stroustrupem ve společnosti AT&T Bell Labs jako rozšíření původního jazyka C. Do jazyka C++ byla oproti původnímu jazyku C přidána objektovost, která byla inspirována jazykem Simula67. Tuto objektovost lze však zcela ignorovat a psát čistě procedurální programy, což z C++ činí multiparadigmatický jazyk. Ačkoli je C++ rozšířením C nejsou tyto dva jazyky zcela kompatibilní. Některé programy jazyka C nelze přeložit za pomoci překladače C++.

Ukázka syntaxe jazyka:

```
#include 'std_lib_facilities.h'

/* Hello World program */

int main()
{
cout << 'Hello World!\n';
return 0;
}
```

Testované překladače:

- Microsoft Visual C++
- G++
- Intel C++

4.3 C#

Programovací jazyk C# (Vyslovované anglicky C Sharp) je moderní, multiparadigmatický, hybridní jazyk, který byl vyvinut v roce 2002 společností Microsoft. Jazyk C# vznikl z jazyků C++, Java a C, ze kterého také převzal syntaxi. Zdrojový kód jazyka C# se nejdříve přeloží do mezikódu nazývaného Microsoft Intermediate Language (MSIL) a poté při spuštění programu je teprve přeložen do podoby strojového kódu. To má za následek nutnost nainstalování překladače na počítač, kde chceme kód spustit.

Ukázka syntaxe jazyka:

```
// Hello World program

public class HelloWorld
{
   public static void Main()
   {
      System.Console.WriteLine("Hello World!");
   }
}
```

Testované překladače:

- Microsoft Visual C#
- Mono C# Compiler

4.4 Java

Java je vysokoúrovňový, multiplatformní a v současné době nejpoužívanější programovací jazyk. Byla vyvinuta v roce 1995 Jamesem Goslingem ve společnosti Sun Microsystems. Java je mylně mnohými považována za čistě objektový jazyk, ale v zájmu efektivity byly její datové typy rozděleny na primitivní a objektové. Java je hybridní jazyk. Překladač nejdříve přeloží zdrojový kód do podoby takzvaného bytekódu, který je následně za běhu interpretován virtuálním strojem Javy, který musí být nainstalován na cílovém zařízení.

Ukázka syntaxe jazyka:

```
/** Hello world program*/

public class HelloWorldApp {

    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Testované překladače:

- Eclipse Compiler for Java
- GNU Compiler for Java
- Oracle Compiler (Javac)

4.5 Object Pascal

Object Pascal vytvořil Niklaus Wirth v roce 1985 pro společnost Apple Computer jako objektové rozšíření jazyka Pascal. Jedná se o kompilovaný vysokoúrovňový jazyk.

Ukázka syntaxe jazyka:

```
/* Hello world program*/

program HelloWorld;
begin
  writeln ('Hello world.')
end
```

Testované překladače:

- Delphi
- Free Pascal

4.6 Scala

Scala je multiparadigmatický programovací jazyk navržený tak, aby spojoval rysy objektově orientovaného a funkcionálního programování. Scala je stejně jako Java nejprve kompilována do bytekódu a teprve poté interpretovaná. Díky tomu může Scala běžet na virtuálním stroji Javy.

Ukázka syntaxe jazyka:

```
/* Hello world program*/

object HelloWorld {
    def main(args: Array[String]) =
        println("Hello World!")
}
```

Testované překladače:

- Scala compiler

5 Vybrané překladače

Tato kapitola je věnovaná překladačům, které byly vybrány pro testování. U každého překladače je krátký popis, použitá verze a odkaz na jeho stránky.

5.1 GCC / G++ 5.3

gcc.gnu.org

Zkratka GCC dříve znamenala GNU C Compiler, ale protože se z GCC stala kolekce překladačů, změnil se význam zkratky na GNU Compiler Collection. Jedná se o kolekci překladačů pro programovací jazyky C, C++, Objective-C, Fortran, Java, Ada a Go. Jedná se o jeden z nejznámějších a nejpoužívanějších překladačů pro jazyk C. GCC je součástí většiny linuxových distribucí.

5.2 Microsoft Visual C++ 2015

visualstudio.com

Překladač pro jazyky C a C++, který je zdarma přímo od společnosti Microsoft. Součástí software je také integrované vývojové prostředí.

5.3 Intel C++

software.intel.com/en-us/c-compilers

Překladač pro programovací jazyky C a C++ od společnosti Intel. Jedná se o placený překladač. Mezi jeho hlavní výhody patří schopnost optimalizace programů pro procesory Intel.

5.4 Mono 4.4

mono-project.com/download

Jedná se o open-source, multiplatformní překladač pro jazyk C#.

5.5 Microsoft Visual C# 2015

visualstudio.com

Překladač pro jazyk C#, který je zdarma přímo od společnosti Microsoft. Součástí software je také integrované vývojové prostředí.

5.6 Oracle Compiler (Javac)

www.oracle.com/technetwork/java/javase/downloads

Překladač pro jazyk Java od společnosti Oracle. Pro testování byl vybrán překladač Javac, který je obsažený v Java JDK 8.

5.7 Eclipse Compiler

www.eclipse.org/jdt/core/

Eclipse Compiler (ECJ) je používán uvnitř IDE Eclipse. ECJ je možné taky použít jako dávkový překladač z příkazové řádky. Pro testy byla použita verze ECJ 4.3.1.

5.8 GNU Compiler for Java (GCJ) 5.3

gcc.gnu.org/java

GCJ je překladač jazyka Java, který je součástí kolekce GNU Compiler Co-

lection. Specialitou této implementace je, že umožňuje překlad jak do bajtkódu, tak i do nativní kódu. Tato možnost ale není v této práci testována.

5.9 Free Pascal 3.0.0

www.freepascal.org

Free Pascal je jeden z nejznámějších open-source překladačů jazyka Object Pascal, který se stále vyvíjí. Free Pascal je k dostání pro mnoho platforem a operačních systémů.

5.10 Delphi

www.embarcadero.com/products/delphi

Delphi je překladač jazyka Object Pascal, který je součástí IDE RAD Studio. Jedná se o placený překladač, ale na oficiálních webových stránkách je možné stáhnout trial verzi programu.

5.11 Scala Compiler 2.11.8

www.scala-lang.org/download/install.html

Scala Compiler je zdarma dostupný překladač pro programovací jazyk Scala.

6 Testování

Testování je důležitou částí vývoje každého softwaru, tedy i překladače. A právě na testování výkonu překladačů se zaměřujeme v této práci. Postup testování lze charakterizovat následujícími termíny. Testování černé skříňky (black-box testing) je způsob testování, při kterém tester pouze ví co má předložený software dělat - dovnitř skříňky se podívat nemůže a neví, jak pracuje. Tester pouze zadá údaje na vstup, a na výstupu dostane odpovídající výsledek. Oproti tomu při testování bílé skříňky (white-box testing) má tester přístup ke zdrojovému kódu softwaru a jeho znalost mu může pomoci při výběru dat, které povedou k chybě programu. Dalšími pojmy jsou takzvané statické testování a dynamické testování. Při statickém testování se testuje program, který neběží. To znamená, že zkoumaný objekt pouze prohlížíme a kontrolujeme. U dynamického testování je testován spuštěný program a práce s ním za pomoci testovacích dat. [2]

Pro naše testování použijeme metodu dynamického testování černé skříňky. Nejdříve vytvoříme testovací vzorky implementací vhodných algoritmů ve zvolených programovacích jazycích, a to tak, aby jejich implementace byla co nepodobnější a neposkytovala žádnému programovacímu jazyku výhodu. Poté testovací vzorky přeložíme za pomoci překladačů vybraných k otestování a budeme měřit hodnoty dle zvolených testovacích kritérií.

6.1 Testovací procedura

Před zahájením samotného testování je nutné určit přesný postup, jak bude testování probíhat. Následující seznam popisuje testovací proceduru použitou při vytváření této práce.

- Definovat cíle testování.
- Určit testovací kritéria
- Určit metody testování
- Implementace testovacích vzorků

- Připravit vhodné PC pro testování
- Získat a nainstalovat testované překladače
- Naměřit hodnoty
- Vyhodnotit naměřené hodnoty
- V případě potřeby opakovat měření
- Ukončit testování a interpretovat výsledky

6.2 Hodnotící kritéria

6.2.1 Doba překladu

Doba překladu je čas, za který překladač převede zdrojové soubory testovacího vzorku na zkompilované spustitelné soubory. Doba překladu by měla být co nejmenší a může se značně lišit v závislosti na použitém překladači a úrovni optimalizace.

6.2.2 Paměťová náročnost přeloženého programu

Paměťová náročnost programu určuje, kolik operační paměti RAM (Random Access Memory) potřebuje přeložený program pro svůj běh. V operační paměti jsou uloženy instrukce programu a data, se kterými program pracuje. Operační paměť je omezená, a proto by ji měl program využívat co nejmenší množství.

6.2.3 Velikost přeloženého programu

Velikost přeložené programu udává, kolik místa zabírá přeložený program na HDD (Hard Disc Drive). Velikost přeloženého programu je silně závislá na použité optimalizaci. Většina kompilátorů nabízí možnost volby zda, chceme optimalizovat pro rychlost běhu programu nebo velikost programu. Místo na HDD je opět omezené, a proto by program měl být co nejmenší.

6.2.4 Rychlost běhu přeloženého programu

Patrně nejdůležitější kritérium. Udává, jak dlouho programu trvá, než provede všechny své instrukce a doběhne do konce.

6.3 Testovací vzorky

Jako testovací vzorky byly vybrány algoritmy, které otestují výkon překladačů v různých úlohách s různými datovými typy a s různě těsnou vazbou na architekturu procesoru. Vzorky byly implementovány co nejpodobněji ve všech testovaných jazycích, tak aby žádný jazyk nebyl zvýhodněn. Například u algoritmů v Javě musel být použit cyklus pro nulování pole i když není potřeba, aby javovské překladače nebyly zvýhodněné například oproti překladačům jazyka C.

Zdrojové soubory všech testovacích vzorků, stejně jako přeložené verze souborů všemi testovanými překladači, se nachází na přiloženém DVD.

Seznam testovacích vzorků:

- Backtracking - Algoritmus pro výpočet všech permutací slova za pomoci backtrackingu.
- Fannkuch - Algoritmus navržený jako benchmark pro překladače.
- GEM - Algoritmus pro výpočet řešení soustavy rovnic za pomoci Gaussovy eliminační metody.
- KMP - Algoritmus pro vyhledání řetězce v textu.
- Mandelbrot - Algoritmus pro výpočet Mandelbrotovy množiny.
- Matrix multiplication - Algoritmus pro vynásobení dvou matic.
- Sieve - Eratosthenovo síto je jednoduchý algoritmus pro nalezení všech prvočísel nižších, než je daná horní mez.
- Spectral norm - Algoritmus pro výpočet spektrální normy matice.

6.4 Testovací prostředí

Testovací prostředí přestavuje softwarovou a hardwarovou sestavu, na které běží testovaný software. Testy byly prováděny na následujících dvou různě výkonných testovacích prostředích.

Testovací prostředí číslo 1

- Procesor: Intel Core i7-3610QM CPU 2.30GHz (8 CPUs)
- Operační systém: Windows 10 Home 64bit, Ubuntu 14.04
- RAM: 8192 MB

Testovací prostředí číslo 2

- Procesor: AMD Turion(tm) 64 X2 Mobile Technology TL-60 2.0GHz (2 CPUs)
- Operační systém: Windows 10 Home 64bit, Ubuntu 14.04
- RAM: 2048 MB

6.5 Využité nástroje

Pro testování rychlosti překladu a rychlosti běhu jednotlivých přeložených programů byl na OS Linux použit program time, který slouží pro změření doby vykonávaného příkazu. Pro spuštění programu time stačí pouze v shellu napsat klíčové slovo time před požadovaný příkaz. Na OS Windows byl pro měření časů použit PowerShell a jeho Cmdlet Measure-Command, který stejně jako program time udává dobu provádění zadaného příkazu. Pro spuštění Cmdletu Measure-Command stačí v PowerShellu zadat Measure-Command a do složených závorek zadat požadovaný příkaz. Všechny testy byly provedeny třikrát a jejich výsledky následně zprůměrovány, aby měření bylo co nejpřesnější.

Pro měření paměťové náročnosti programů byly v obou systémech použity jejich standardní nástroje. U OS Windows byl pro měření použit Správce procesů a u OS Linux byl použit program htop.

Pro zjištění velikosti přeložených programů byl na OS Windows použit průzkumník souborů.

7 Výsledky měření

V této kapitole se zaměříme na vyhodnocení naměřených výsledků a jejich následný rozbor. Výsledky budou rozřazeny dle jednotlivých hodnotících kritérií. Kvůli lepší čitelnosti a možnosti porovnání budou data zobrazeny v tabulce.

7.1 Doba překladu

[ms]	**Backtr.**	**Fann.**	**Gem**	**KMP**	**Mandel.**	**MM**	**Sieve**	**Spec. Norm.**
Java								
Javac	525	555	560	540	530	536	529	558
Eclipse	462	479	465	462	456	459	460	466
GCJ	285	286	597	276	286	287	275	278
C								
Visual C++	452	530	536	480	534	502	573	486
Intel C++	703	769	810	783	760	690	692	683
GCC	65	236	75	65	90	76	67	97
C++								
Visual C++	570	530	564	573	564	540	501	504
Intel C++	803	732	843	771	724	683	629	674
G++	87	86	91	86	87	81	87	116
C#								
Visual C#	1362	1542	401	1681	1921	321	1462	1492
Mono	482	465	463	480	462	459	471	481
Object Pascal								
Delphi	1191	723	1122	1421	935	1322	733	2433
Free Pascal	1136	651	1011	1183	1179	1031	586	2121
Scala								
Scala compiler	3189	3735	3876	3933	3542	3655	3234	3754

Tabulka 7.1: Doba kompilace jednotlivých překladačů v milisekundách.

Měření rychlosti překladu bylo bohužel prováděno pouze na prvním testovacím prostředí z důvodu licencí testovaných překladačů.

Z tabulky 7.1 je možné vyčíst, jak dlouho trval překlad testovacích vzorků jednotlivým překladačům. Z tabulky je patrné, že nejrychlejší byl překlad testovacích vzorků napsaných v jazycích C a C++ za pomoci linuxových kompilátorů GCC a G++. Překlad testovacích vzorků těmito překladači trval okolo 90 ms. U windowsového překladače jazyka C Visual trval překlad v průměru 5x déle a u Intel C++ dokonce 7x.

Při překladu testovacích vzorků napsaných v jazyce Java byl linuxový překladač GCJ, který pochází ze stejné skupiny jako překladače GCC a G++, opět rychlejší než windowsové překladače. Rychlost překladu GCJ byla rychlejší o cca 100 ms než překladač Eclipse a o 200 ms než překladač Javac.

Při testování vzorků jazyka C# byl opět rychlejší Linuxový překladač. Kompilátor Mono přeložil testované soubory v průměru o 1 sekundu rychleji než windowsovský překladač Visual C#.

U kompilátorů jazyka Object Pascal byla rychlost překladačů Delphi a Free Pascal srovnatelná.

Překlad testovacích vzorků napsaných v jazyce Scala byl nejpomalejší. Překlad trval průměrně 3600 ms.

7.2 Velikost přeloženého programu

[kB]	**Backtr.**	**Fann.**	**Gem**	**KMP**	**Mandel.**	**MM**	**Sieve**	**Spec. Norm.**
Java								
Javac	0,72	1,57	1,36	1,06	1	0,94	0,725	1,39
Eclipse	0,68	1,56	1,32	1,02	0,99	0,631	0,694	1,32
GCJ	0,68	6,91	1,17	1,02	0,99	0,631	0,694	1,32
C								
Visual C++	9,5	9,5	9,5	9,5	9,5	9,5	9,5	10
Intel C++	27	28	29,5	27,5	27,5	28	27,5	29,5
GCC	8,36	8,71	8,56	8,43	8,5	8,44	8,38	8,61
C++								
Visual C++	9,5	9,5	9,5	9,5	9,5	9,5	9,5	10
Intel C++	27	28	29,5	27,5	27,5	28	27,5	29,5
G++	8,37	8,71	8,56	8,44	8,51	8,44	8,38	8,65
C#								
Visual C#	5	5,5	5,5	5,5	5,5	5,5	5,5	5,5
Mono	3,5	4	4	3,5	3,5	3,5	3,5	4
Object Pascal								
Delphi	1150	1151	1150	1150	1150	1150	1150	1151
Free Pascal	161	161	161	161	161	161	161	161
Scala								
Scala compiler	3,63	14,2	16,2	12,5	12,7	16,4	5,86	16,3

Tabulka 7.2: Velikost výsledných přeložených souborů testovacích vzorků v kB.

Tabulka 7.2 udává velikosti souborů vytvořených překladem testovacích vzorků testovanými překladači. Před provedením měření jsem očekával, že překladače, které byly nejrychlejší při měření rychlosti překladu, budou produkovat největší soubory. Moje domněnka se však nepotvrdila, protože li-

nuxové překladače z rodiny GNU, které byly v minulém testu zdaleka nejrychlejší, produkovaly stejně velké soubory a v některých případech dokonce menší než ostatní překladače. Naopak podle mého očekávání nejmenší soubory vytvářely hybridní jazyky Java a C#.

Nejmenší soubory produkovaly překladače jazyka Java. Všechny 3 testované překladače vytvářely soubory o velikosti přibližně 1 kB.

Druhé nejmenší soubory vytvářely překladače již zmíněného jazyka C#. Překladač Mono vytvářel soubory o přibližné velikosti 4 kB zatímco překladač Visual C++ soubory o cca 1,5 kB větší.

U překladačů jazyků C a C++ byly největší rozdíly mezi překladačem Intel C++ a zbylými dvěma překladači. Překladače GNU a Visual C++ produkovaly srovnatelně velké soubory, zatímco překladač Intel C++ soubory v průměru o 20 kB větší.

Překvapením pro mě byla velikost souborů vytvořených překladačem Scala Compiler. I přesto, že Scala je také hybridní jazyk velikost souborů vytvořených kompilací byla větší než u ostatních hybridních jazyků. Velikost souborů byla v průměru 12 kB.

Největší rozdíl mezi velikostmi souborů jednoho jazyka byl však u jazyka Object Pascal. Zatímco Free Pascal vytvářel soubory o průměrné velikosti 161 kB, překladač Delphi vytvářel soubory o shodné velikosti 1150 kB.

7.3 Doba běhu

Doba běhu přeložených testovacích vzorků byla měřena v obou testovacích prostředích. Výsledky z měření na prvním výkonnějším testovacím prostředí jsou zaznamenány v tabulce 7.3 a výsledky z druhého méně výkoného testovacího prostředí se nalézají v tabulce 7.4. Největší rozdíly mezi dobami běhu byly mezi překladači různých programovacích jazyků. Podle očekávání byly nejrychlejší programy napsané v jazyce C a C++. Cílem této práce je ovšem porovnat výkon překladačů a proto se dále zaměřím na rozdíly času běhu mezi překladači stejných jazyků.

U jazyka C byl na obou testovacích prostředích nejrychlejší překladač Intel C++. Překladač GCC dosahoval velmi rozličných časů. U testovacího

[ms]	Backtr.	Fann.	Gem	KMP	Mandel.	MM	Sieve	Spec. Norm.
Java								
Javac	4041	40142	1799	929	4218	7646	2176	18264
Eclipse	4024	37297	1842	889	5911	7674	2185	18213
GCJ	4344	48512	1661	862	5946	1895	2085	17984
C								
Visual C++	3895	32250	3190	328	3894	7147	1640	17463
Intel C++	2604	28182	1624	527	3614	7171	1583	8740
GCC	3841	30680	3427	418	3726	1253	1680	35648
C++								
Visual C++	3766	32206	3289	318	3891	7161	1585	17463
Intel C++	2567	28232	1686	528	3608	7183	1574	8745
G++	3921	29407	3173	434	3743	1089	1531	35628
C#								
Visual C#	6383	67127	16891	1073	7844	8542	2449	17515
Mono	7875	97910	37688	1340	7336	10585	2702	38063
Object Pascal								
Delphi	15233	70655	11800	991	7872	7227	2411	38221
Free Pascal	10226	76461	10144	984	7589	7323	2319	32523
Scala								
Scala compiler	3128	52245	6722	7904	4737	8347	2791	28144

Tabulka 7.3: Doba běhu testovacích vzorků v milisekundách na prvním testovacím prostředí.

vzorku Matrix Multiplication byl o 6 sekund rychlejší než Visual C++ a Intel C++, avšak například u testovacího vzorku Spectral Norm byl podstatně pomalejší než zbylé dva překladače.

Časy běhu přeložených testovacích vzorků jazyka C++ byly velmi podobné jako časy běhu u jazyka C. Opět byl na obou testovacích prostředích nejrychlejší překladač Intel C++.

U jazyka Java na prvním výkonnějším testovacím prostředí dosahovaly všechny testovací vzorky podobných časů bez ohledu na překladač, který byl použit pro jejich přeložení. Na druhém méně výkonném testovacím prostředí ovšem byl linuxový překladač GCJ mnohem rychlejší než windowsové překladače Javac a Eclipse. Obzvláště u testovacích vzorků GEM, Matrix Multiplication a Spectral Norm byl rozdíl mezi překladačem GCJ a jeho windosovými protějšky propastný.

U jazyka C# byly testovací vzorky přeložené překladačem Visual C# rychlejší než vzorky přeložené za pomoci překladače Mono. Rozdíl mezi těmito překladači byl obzvláště patrný na druhém méně výkonném testovacím prostředí u vzorků Fannkuch a Spectral Norm, kde rozdíl mezi těmito překladači byl více jak jedna minuta.

[ms]	Backtr.	Fann.	Gem	KMP	Mandel.	MM	Sieve	Spec. Norm.
Java								
Javac	12872	160161	31659	2937	15805	26067	7230	166223
Eclipse	13795	149889	31199	3156	15538	26080	7229	167483
GCJ	13727	123916	9963	3320	11815	14664	6640	37680
C								
Visual	9858	55954	15615	763	7919	14274	7146	41740
Intel	8695	55674	11129	722	8429	13851	6236	57966
GCC	10026	55597	14176	1085	8176	3769	4479	42025
C++								
Visual	8881	57662	15344	760	7936	14097	6451	41400
Intel	7783	55523	10604	720	8429	13897	6250	58020
G++	10298	56058	14214	1067	9159	3483	4491	42012
C#								
Visual	18129	191437	49672	6685	15299	29437	8052	40284
Mono	29360	267295	200280	3966	15025	51894	7471	106376
Object Pascal								
Delphi	26628	206858	60515	2833	14545	22204	7457	180772
Free Pascal	30171	212029	48977	2676	15123	21079	7615	163423
Scala								
Scala	17823	105644	105644	16940	50039	45557	11314	193745

Tabulka 7.4: Doba běhu testovacích vzorků v milisekundách na druhém testovacím prostředí.

Testovací vzorky jazyka Object Pascal, které byly přeložené překladači Delphi a Free Pascal dosahovaly velmi podobných časů. U vzorku Fannkuch byl mírně rychlejší překladač Delphi a naopak u vzorku Spectral Norm byl rychlejší překladač Free Pascal.

Vzorky přeložené pomocí překladače Scala Compiler byly ze všech vzorků nejvíce ovlivněny změnou testovacího prostředí. Se sníženým výkonem testovacího prostředí prudce rostl čas běhu programu. Nejpropastnější tento rozdíl byl u vzorku Fannkuch. Na prvním testovacím prostředí s vyšším výkonem trval běh vzorku Fannkuch 52245 milisekund zatímco na druhém méně výkoném testovacím prostředí vzorek Fannkuch běžel 1425346 milisekund.

7.4 Paměťová náročnost přeloženého programu

Doba běhu přeložených testovacích vzorků byla měřena v obou testovacích prostředích. Výsledky z měření na prvním výkonnějším testovacím prostředí jsou zaznamenány v tabulce 7.5 a výsledky z druhého méně výkoného testovacího prostředí se nalézají v tabulce 7.6. Množství využívané paměti bylo velmi podobné v obou testovacích prostředích.

[MB]	Backtr.	Fann.	Gem	KMP	Mandel.	MM	Sieve	Spec. Norm.
Java								
Javac	8.8	8.5	25.4	257.9	8.9	21.5	392.1	9.1
Eclipse	8,7	8.6	24.9	257.9	8.9	20.8	391.8	8.8
GCJ	23.4	26.3	42.7	282.9	25.3	38.7	418.4	25.7
C								
Visual C++	0.6	0.5	15.8	143.6	0.5	14.4	382	0.9
Intel C++	0.8	0.6	18.9	143.6	0.5	14.3	382	0.9
GCC	0.6	0.6	16.3	148.6	0.6	12.4	391.3	0.9
C++								
Visual C++	0.6	0.5	15.8	143.6	0.5	14.4	382	0.9
Intel C++	0.6	0.6	18.9	134.6	0.6	14.4	382	0.9
G++	0.6	0.6	16.2	148.4	0.6	12.5	394.8	1.4
C#								
Visual C#	1.3	1.3	16.6	249.4	1.3	12.8	382.8	1.5
Mono	13.1	13	28.6	267.3	13.2	24.7	403.7	13.2
Object Pascal								
Delphi	0.7	0.8	31.3	248.7	0.8	12.2	382.2	1.1
Free Pascal	0.9	0.9	31.4	143.5	0.8	12.3	382.3	1.0
Scala								
Scala compiler	109	48.9	62.3	695.3	52.3	50.7	439.1	48

Tabulka 7.5: Paměťová náročnost přeložených programů v megabytech na prvním testovacím prostředí.

U kompilovaných jazyků byla C a C++ byla paměťová náročnost testovaných vzorků přibližně stejná bez ohledu na použitý kompilátor.

U hybridního jazyka Java byl rozdíl ve využité paměti mezi linuxovými a windowsovými překladači. Zatímco vzorky přeložené za pomoci windowsových překladačů Javac a Eclipse využívali přibližně stejné množství paměti, vzorky přeložené za pomoci linuxovového překladače GCJ využíval průměrně o 20 MB více.

U hybridního jazyka C# byly výsledky podobné jako v případě jazyka Java. Vzorky přeložené za pomoci linuxového překladače Mono využívaly průměrně o 15 MB více než vzorky přeložené za pomoci překladače Visual C#.

Object pascalové testovací vzorky přeložené za pomoci překladače Delphi využívaly velmi podobné množství paměti jako testovací vzorky přeložené za pomoci překladače Free Pascal. Jediný rozdíl nastal v případě testovacího vzorku KMP, kde program přeložený za pomoci překladače Delphi využil o 105 MB paměti více než program přeložený za pomoci překladače Free Pascal.

Nejvíce paměti ze všech měřených vzorků spotřebovávaly testovací vzorky

[MB]	**Backtr.**	**Fann.**	**Gem**	**KMP**	**Mandel.**	**MM**	**Sieve**	**Spec. Norm.**
Java								
Javac	4.9	4.9	22.1	254	4.9	17.7	388.4	5.1
Eclipse	4.9	4.9	22.1	254	4.9	17.8	388.4	5.1
GCJ	22.1	23.2	38.1	276.7	22.1	38	412.9	23.3
C								
Visual C++	0.3	0.4	15.6	143.6	0.3	11.7	382	0.6
Intel C++	0.4	0.3	15.6	143.6	0.4	17.8	382	0.5
GCC	0.5	0.5	16.2	147	0.5	12.2	391.2	0.7
C++								
Visual C++	0.4	0.4	15.6	143.6	0.3	11.8	381.6	0.6
Intel C++	0.4	0.4	15.6	143.6	0.4	11.8	381.6	0.6
G++	0.5	0.5	16.1	147	0.5	12.3	391.2	0.7
C#								
Visual C#	1.5	1.6	16.9	249	1.6	13.1	383.1	1.8
Mono	12	12	24.4	268.1	12.1	24.4	403.4	12.9
Object Pascal								
Delphi	0.5	0.6	31.3	248.5	0.5	12	382	0.8
Free Pascal	0.6	0.6	31.1	143.6	0.6	12.1	382.1	0.8
Scala								
Scala compiler	109	48.9	62.3	695.3	52.3	50.7	439.1	48

Tabulka 7.6: Paměťová náročnost přeložených programů v megabytech na druhém testovacím prostředí.

přeložené za pomoci překladače Scala Compiler.

8 Shrnutí výsledků

V předchozí kapitole byly zobrazeny výsledky měření dle daných hodnotících kategorií a následně okomentovány. Tato kapitola se bude bude věnovat shrnutí poznatků získaných měřením a vyzdvihnutí nejlepších překladačů každého jazyka.

8.1 C

Mezi testovanými překladači jazyka, kterými byly Visual C++, Intel C++ a GCC, bych označil jako nejlepší překladač Intel C++. I přes to, že GCC dosahovalo zdaleka největší rychlosti při překladu programů, tak bych jako nejlepší překladač pro C označil Intel C++, jehož vzorky měly ve většině případů nejkratší dobu běhu programu.

8.2 C++

Naměřené hodnoty pro jazyk C++ se velmi podobaly naměřeným výsledkům u jazyka C, a proto bych ze stejných důvodů jako u jazyka C zvolil jako nejlepší překladač Intel C++.

8.3 C#

U jazyka C# probíhalo měření na překladačích Visual C# a Mono. Překlad programů byl u překladače Visual C# v průměru o sekundu pomalejší a jeho přeložené programy o 1 kB větší než u překladače Mono, ale měly mnohem lepší rychlost běhu programu a využívaly méně operační paměti. Rychlost běhu programu a paměťovou náročnost programů pokládám za důležitější kritéria než velikost programů a rychlost překladu, a proto bych jako lepší překladač zvolil Visual C#.

8.4 Java

Překladače Eclipse a Javac měly velmi podobné výsledky ve všech testech zatímco překladač GCJ byl vždy o něco horší. Jediným testem, ve kterém zvítězil překladač GCJ, byl test rychlosti překladu. Dobu překladu nepokládám za tak důležité kritérium, a proto bych pro překlad javovských programů volil překladač Eclipse nebo Javac.

8.5 Object Pascal

Pascalovské překladače Delphi a Free Pascal dosahovaly v testech velmi podobných výsledků. Pokud bych vzal v úvahu vysokou cenu licence překladače Deplhi, tak bych pro překlad pascalových programů volil překladač Free Pascal.

8.6 Scala

Jediným testovaným překladačem jazyka Scala byl překladač Scala Compiler. Tento překladač bych označil jako nejhorší ze všech měřených překladačů. Překlad trval Scala Compileru nejdéle ze všech měřených překladačů, programy přeložené tímto překladačem využívaly nejvíce paměti a na nevýkonném testovacím prostředí byla doba běhu programů přeložených tímto překladačem velmi dlouhá.

8.7 Nejlepší překladač

Jako celkový nejlepší překladač bych označil Intel C++, protože dokáže překládat programy v jazyce C i C++, programy přeložené tímto překladačem mají nejkratší dobu běhu a využívají nejméně operační paměti.

9 Závěr

V rámci této bakalářské práce byla popsána teorie programovacích jazyků, struktura a funkce překladačů programovacích jazyků, informace o zvolených jazycích a překladačích, metody pro měření výkonu programovacích jazyků a byly vybrány jazyky a překladače pro měření. Pro měření bylo vybráno šest programovacích jazyků a dvanáct překladačů. Vybranými jazyky byly C, C++, C#, Java, Object Pascal a Scala. Pro otestování byly vybrány překladače Gcc, Microsoft Visual C++, Intel C++, G++, Eclipse, Javac, GCJ, Microsoft Visual C#, Mono, Delphi, Free Pascal a Scala Compiler.

Následně byly implementovány testovací vzorky, na kterých bylo provedeno měření. Výsledky byly prezentovány, zhodnoceny a na jejich základě byl vybrán nejlepší překladač pro každý z testovaných jazyků.

Literatura

[1] ŠIMEčEK, P. *Formální jazyky a automaty [on-line].* Dostupné z: `https://mks.mff.cuni.cz/library/FormalniJazykyAAutomatyPS/FormalniJazykyAAutomatyPS.pdf`.

[2] PATTON, R. *Testování softwaru.* : Computer Press, 2002. ISBN 80-7226-636-5.

[3] VAVREčKOVá. *Programování překladačů.* : Ediční středisko FPF SU v Opavě, 2008. ISBN 978-80-7248-493-5.

[4] SKOUPIL, D. *Úvod do paradigmat programování [on-line]*, 2007. Dostupné z: `https://phoenix.inf.upol.cz/esf/ucebni/uvod_para.pdf`.

[5] TREMBLAY, J.-P. *The Theory and Practice of Compiler Writing (McGraw-Hill computer science series).* : Mcgraw-Hill College, 1985. ISBN 978-0070651616.

[6] VYCHODIL, V. *PARADIGMATA PROGRAMOVÁNÍ 1A [on-line].* KATEDRA INFORMATIKY, PŘÍRODOVĚDECKÁ FAKULTA, UNIVERZITA PALACKÉHO, 2008. Dostupné z: `https://phoenix.inf.upol.cz/esf/ucebni/pp1a.pdf`.

Přílohy

A Obsah přiloženého CD

- Testovací vzorky
  - Složka obsahuje testovací vzorky použité pro měření výkonu překladačů. Testovací algoritmy jsou nejdříve rozčleněny do složek podle jazyku v jakém byly implementovány a poté podle názvu algoritmů. Ve složce src se vždy nachází zdrojové soubory a přeložené soubory jsou vždy umístěny ve složce s názvem překladače, který byl použit pro jejich překlad.
- Dokumentace
  - text – PDF soubor s textem bakalářské práce,
  - zdroj – zdrojové soubory TeXu bakalářské práce a obrázky.