

Podpora jazyka Monkey C v prostředí VS Code

Monkey C Language Support in VS Code

Jakub Pšenčík

Bakalářská práce

Vedoucí práce: Ing. Jan Janoušek

Ostrava, 2021

Abstrakt

V této bakalářské práci se budu zabývat vývojem rozšíření pro Visual Studio Code, jenž bude poskytovat plnou podporu jazyka Monkey C. Součástí práce bude představení prostředí Visual Studio Code a problematika vývoje rozšíření v tomto prostředí. Představen bude, mimo jiné, také jazyk Monkey C. V další části práce se budu zabývat nástrojem ANTLR, díky kterému jsme schopni generovat vlastní překladač jazyka pomocí bezkontextové gramatiky, parsováním kódu a jeho syntaktickou analýzou. Závěrem bude výsledné rozšíření testováno.

Klíčová slova

bakalářská práce, rozšíření, Monkey C, Typescript, parser

Abstract

In this bachelor thesis I will deal with the development of extension for Visual Sstudio Code, which will provide full support for Monkey C language. The work will include an introduction to the Visual Studio Code environment and the development of extensions in this environment. Among other things, the Monkey C language will be introduced. In the next part of the work I will describe the ANTLR tool, used to generate our own language compiler using context-free grammar, code parsing and its syntactic analysis. Finally, the resulting extension will be tested.

Keywords

bachelor thesis, extension, Monkey C, Typescript, parser

Poděkování

Rád bych na tomto místě poděkoval svému vedoucímu práce Ing. Janu Janouškovi za odborné a metodické vedení, ochotu a pomoc v průběhu vypracovávání práce.

Obsah

Seznam použitých symbolů a zkratk	6
Seznam obrázků	7
Seznam tabulek	8
1 Úvod	9
2 Jazyk Monkey C	10
3 Problematika vývoje rozšíření pro VS Code	13
3.1 Visual Studio Code	13
3.2 Typescript	14
3.3 Komponenty potřebné pro tvorbu rozšíření	14
3.4 ANTLR - Nástroj pro generování překladače	14
4 Syntaktická a sémantická analýza kódu	18
4.1 Parser a Lexer	18
4.2 Syntaktický strom	20
4.3 ANTLR Listener a callback funkce	22
5 Návrh a implementace rozšíření	24
5.1 Error Listener	24
5.2 Modul Toybox	25
5.3 Zvýraznění kódu	25
5.4 Automatické doplňování a našeptávání kódu	25
5.5 Nedostatky rozšíření	27
6 Testování výsledného řešení	29
7 Závěr	30

Seznam použitých zkratek a symbolů

ANTLR	– ANother Tool for Language Recognition
AST	– Abstract syntax tree
VS Code	– Visual Studio Code
API	– Application Programming Interface

Seznam obrázků

2.1	ukázka jednoduchého fragmentu kódu v MonkeyC	11
2.2	rozšíření při pokusu předat funkci, jako parametr, detekuje chybu.	12
3.1	ukázka hlavičky gramatiky MonkeyC.g4 pro popis jazyka	15
3.2	potřebné soubory vygenerované nástrojem ANTLR	16
3.3	"ParseTreeInspector"- vizuální podoba syntaktického stromu	17
4.1	přiřazení hodnoty 1 proměnné input	19
4.2	Ukázka, jak Language recognizer zpracovává vstupní sekvenci znaků	19
4.3	funkce vygenerované nástrojem ANTLR	23
5.1	ukázka neobarveného kódu v MonkeyC	26
5.2	ukázka obarvení kódu v MonkeyC	26
5.3	ukázka automatického našeptávání kódu na proměnné typy string	28
5.4	komentář nad funkcí obsahující stručný popis, parametry funkce a návratový typ . .	28
5.5	nedostatek rozšíření	28
5.6	chybová hláška z error listeneru	28

Seznam tabulek

Kapitola 1

Úvod

V dnešní době, kdy jsme obklopeni spoustou moderních technologií, existující programovací jazyky se stále vyvíjejí kupředu a nové postupně vznikají, existuje nespočet nástrojů, rozšíření, vývojových prostředí, které práci a vývoj v těchto jazycích dokážou v mnoha ohledech usnadnit. Jazyk Monkey C, jenž bude středobodem této práce, zatím nedisponuje tak široce obsáhlou komunitou, jako mají např. v dnešní době velmi populární Javascript, Python, C Sharp, Ruby, atd...

Typickým příkladem společnosti, která se zabývá právě vývojem softwarů pro programátory či vývojáře, je česká JetBrains s.r.o. Ti mají na kontě již mnoho produktů usnadňujících programování, např. ReSharper (.NET), IntelliJ IDEA (Java, Groovy, atd...) či PyCharm (Python). Ovšem na podporu Monkey C zatím žádné softwarové řešení v JetBrains do světa nepustili. [1]

Na oficiálním webu, kde Microsoft nabízí nejrozšířenější rozšíření [2], je sice možné nalézt několik nástrojů, které s vývojem v Monkey C pomáhají. Cílem této práce je však tvorba rozšíření, které poskytne uživateli co největší podporu a hlavně, aby byla postavena na vlastním řešení. Následujících kapitoly tedy budou věnovány tvorbě a vývoji rozšíření pro vývojové prostředí Visual Studio Code, které poskytne plnou podporu při vývoji aplikací v Monkey C. V kapitole 2 dojde k představení samotného jazyka Monkey C a operačního systému Connect IQ, na kterém Monkey C běží. V kapitole 3 jsou popsány veškeré nástroje a komponenty, které budou při tvorbě rozšíření použity. Pro tvorbu rozšíření bude využit jazyk Typescript, dále pak nástroj ANTLR, který je schopen vygenerovat překladač jazyka. Toho je schopen dosáhnout za pomoci jeho popisu obsaženého v bezkontextové gramatice, jenž byla pro tuto práci poskytnuta, a tudíž její vytváření nebylo součástí práce. Kapitola 4 se popisuje analýzu kódu jak z pohledu syntaxe, tak sémantiky. V kapitole 5 se práce zabývá již samotným návrhem a implementací rozšíření. Objeví se zde témata, jako parsování kódu, automatické doplňování a našptávání kódu, atd... Kapitola 6 je poté věnována testování rozšíření.

Kapitola 2

Jazyk Monkey C

Monkey C [3] je objektově orientovaný jazyk. Byl navržen americkou společností Garmin Ltd. [4] a jeho hlavním účelem je snadný vývoj Connect IQ aplikací pro nositelná zařízení (nejvhodnějším příkladem jsou zde chytré hodinky). Jedná se o dynamický programovací jazyk, podobně jako Java, PHP či Ruby. Z těchto uvedených jazyků také Monkey C vychází. Cílem Monkey C je zjednodušit proces vytváření samotné aplikace a umožnit tak vývojářům více se soustředit na zákazníka a méně na omezení zdrojů. Využívá tzv. "reference counting" k automatickému čištění paměti, což vývojáře osvobozuje od manuální správy paměti (např. jako v jazyce C/C++).

Všechny aplikace, naprogramované v Monkey C, běží na operačním systému (vývojářské platformě) Connect IQ. [5] Connect IQ umožňuje jak jednotlivcům, tak velkým společnostem vyvíjet nespočet různých aplikací a rozšíření pro Garmin zařízení. Můžeme je tedy zařadit mezi takové platformy, jako jsou Android či iOS. S pomocí Connect IQ je tedy možné vyvíjet např.:

1. **aplikace** - jedná se o plně funkční aplikace běžící na hodinkách. Může se jednat např. o hudební aplikaci, která synchronizuje obsah s mobilní aplikací na zařízení uživatele (např. Spotify, iTunes, atd...)
2. **ciferníky** (Watch Faces) - ciferník si lze představit, jako domovskou obrazovku na telefonu, které uživateli dokáže zobrazit celou řadu informací (záleží samozřejmě na preferencích konkrétního jedince). Ciferník je na hodinkách aktualizován každých 60 vteřin a běží nepřetržitě v režimu nízké spotřeby.
3. **widgety** - opět se jedná o komponentu, která je běžně využívána na mobilních zařízeních. Widget dokáže, za pomoci dat z hodinek nebo připojeného telefonu, zobrazovat nejrozličnější informace od počasí, přes puls uživatele až po notifikace příchozích hovorů.

Na obrázku 2.1 je možnost vidět jednoduchý fragment kódu v Monkey C. Z obrázku je patrné, že Monkey C, stejně jako většina dnešních programovacích jazyků, podporuje operace, jako dědičnost, určení rozsahu jmenného prostoru pomocí klíčového slova `using`, a mnoho dalších. "Stejně jako je

```

using Toybox.Application as App;
using Toybox.System;

class MyProjectApp extends App.AppBase {

    // onStart() is called on application start up
    function onStart(state) {
    }

    // onStop() is called when your application is exiting
    function onStop(state) {
    }

    // Return the initial view of your application here
    function getInitialView() {
    return [ new MyProjectView() ];
    }
}

```

Obrázek 2.1: ukázka jednoduchého fragmentu kódu v MonkeyC

tomu v programovacím jazyce Java, Monkey C kompiluje zdrojové soubory do byte kódu, který je následně interpretován virtuálním strojem Monkey Brains. Virtuální stroj Monkey Brains poté komunikuje s dalšími dostupnými API.[6] Tyto API slouží např. pro práci s polohou zařízení, komunikací se senzory detekující nejrůznější informace (teplotu ovzduší, tlak vzduchu, puls, atd...).

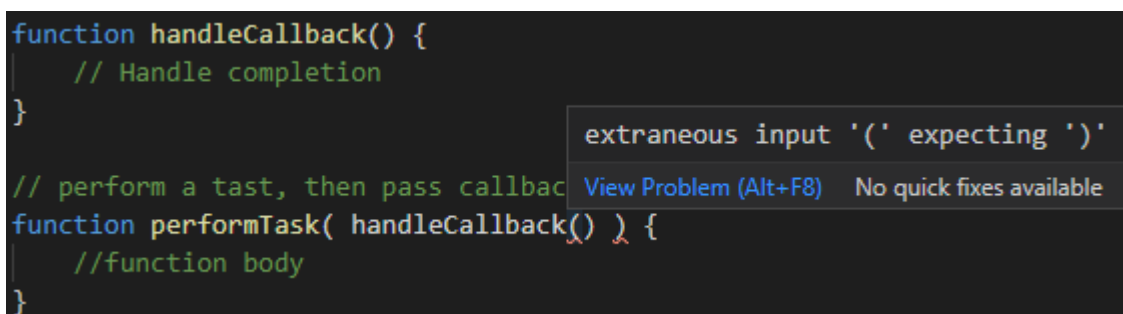
2.0.1 Monkey C a ostatní jazyky

Stejně jako italština a španělština vznikly z latiny, Monkey C čerpá spoustu věcí převážně z jiných moderních jazyků. C, Java, JavaScript, Python, Lua, Ruby a PHP, všechny tyto jazyky ovlivnily výslednou podobu Monkey C. [3]

Mezi Monkey C a výše uvedenými jazyky jsou přesto jisté rozdíly, které budou popsány v následujících řádcích.

1. **Java** - Stejně, jako Java, je Monkey C kompilován do byte kódu, který je následně interpretován virtuálním strojem. Jak už bylo zmíněno výše, virtuální stroj pro Monkey C nese název Monkey Brains. Další společná vlastnost s Javou je ta, že k alokaci objektů dochází na haldě (heap). V neposledí řadě stojí za zmínku, že virtuální stroj má na starosti čištění paměti, přičemž v Javě je toto realizováno prostřednictvím tzv. "garbage collectoru" a v Monkey C pomocí "reference countingu".

2. **JavaScript** - Hlavním rozdílem mezi JavaScriptem a Monkey C spočívá v tom, že funkce v Monkey C nedisponují vlastností "funkce první třídy" (first-class function). "To znamená, že jazyk podporuje předávání funkcí jako argumentů jiným funkcím, jejich vrácení jako hodnoty z jiných funkcí a jejich přiřazení k proměnným nebo jejich uložení v datových strukturách." [7] Na obrázku 2.2 lze vidět, že při pokusu předat funkci, jako parametr jinou funkci, rozšíření detekuje operaci, která je v rozporu s Monkey C gramatikou, a tím pádem oznámí uživateli chybu.
3. **Python** - Objekty v Pythonu jsou reprezentovány, jako hashovací tabulky, přičemž funkce a proměnné lze objektům přiřazovat za běhu programu. Objekty v Monkey C jsou kompilovány ještě před spuštěním programu, a tím pádem nemohou být modifikovány za běhu. Všechny proměnné, předtím než je použijeme např. ve funkci, třídní instanci či rodičovském modulu, musí být deklarovány.



```
function handleCallback() {  
    // Handle completion  
}  
  
// perform a task, then pass callback  
function performTask( handleCallback() ) {  
    //function body  
}
```

extraneous input '(' expecting ')'
[View Problem \(Alt+F8\)](#) No quick fixes available

Obrázek 2.2: rozšíření při pokusu předat funkci, jako parametr, detekuje chybu.

Kapitola 3

Problematika vývoje rozšíření pro VS Code

Jak už bylo zmíněno v úvodní kapitole, na oficiálním webu s rozšířeními pro VS Code [2] najdeme spousty aplikací, které vývojářům pomáhají např. s "debuggováním" kódu, analýzou dat, strojovým učením (machine learning), atd... Rozšíření pro Monkey C však nejsou zastoupena v tak hojném počtu, jako ostatní jazyky, což lze vidět v tabulce níže 3.1. V této kapitole budou popsány komponenty, které jsou stěžejní pro vytvoření překladače jazyka, parseru jazyka a následně rozšíření samotného.

3.1 Visual Studio Code

Visual Studio Code [8] je editor zdrojového kódu vytvořený společností Microsoft pro operační Windows, Linux a MacOS, který podporuje stovky druhů jazyků. Je naprogramován v jazycích JavaScript a Typescript a nabízí spoustu užitečných funkcí, mezi které patří podpora ladění kódu, zvýrazňování syntaxe, automatické doplňování a našeptávání kódu, odsazování textu, intuitivní klávesové zkratky, které usnadňují navigaci v kódu, atd... Cílem práce je integrovat většinu těchto funkcí a možností do finálního rozšíření.

Tabulka 3.1: počet výsledků po vyhledání daného jazyka na marketplace [2]

název jazyka	počet výsledků
C Sharp	169
Java	2521
Python	454
R	6350
Monkey C	2

3.2 Typescript

TypeScript [9] je open-source programovací jazyk vyvinutý společností Microsoft. Jedná se o nádstavbu nad jazykem JavaScript, která jej rozšiřuje o statické typování a další atributy, které známe z objektově orientovaného programování jako jsou třídy, moduly a další. Samotný kód psaný v jazyce TypeScript se kompiluje do jazyka JavaScript. Jelikož je tento jazyk nádstavbou nad JavaScriptem, je každý JavaScript kód automaticky validním TypeScript kódem.

S programovacím jazykem Typescript jsem na začátku této práce mnohem menší zkušenosti, než např. s programovacími jazyky C Sharp nebo JavaScript. V průběhu studia jsem se však setkal s JavaScriptem při vývoji internetových aplikací. A jelikož je Typescript pouze nádstavba JavaScriptu a objektově orientované programování mě bylo dobře známé z jazyka C Sharp, nebylo obtížné se veškeré chybějící potřebné znalosti rychle doučit.

3.3 Komponenty potřebné pro tvorbu rozšíření

Předtím, než bude možné začít pracovat na samotném rozšíření, je potřeba obstarat několik důležitých nástrojů a komponent, jenž jsou klíčové při vývoji. Nejdůležitějším nástrojem pro vývoj rozšíření je však ANTLR, který je popsán v sekci 3.4.

3.3.1 Gramatika

Jako první je potřeba definovat popis jazyka Monkey C. V tomto případě je jazyk Monkey C popsán prostřednictvím bezkontextové gramatiky, která formálně definuje syntax (pravidla) jazyka. Jedná se, ve své podstatě, o soubor pravidel, kde každé pravidlo reprezentuje určitou strukturu či frázi jazyka. Z tohoto formálního popisu je ANTLR schopen vygenerovat parser daného jazyka, který dokáže automaticky sestavit datovou strukturu, která se nazývá buďto "parse tree" nebo "syntax tree", což v češtině znamená **syntaktický strom**. Tento syntaktický strom poté reprezentuje, jak přesně je gramatika schopna rozpoznávat vstupní data (např. fragment kódu).

Jelikož je v této práci použit ANTLR verze 4, je klíčové, aby název souboru obsahující gramatiku končil příponou .g4, v tomto případě soubor nese název **MonkeyC.g4**.

Na obrázku 3.1 lze vidět prvních pár řádků Monkey C gramatiky. Gramatika obsahuje spoustu známých klíčových slov, nebo-li tokenů, např. "CLASS", "FUNCTION", "USING", atd...

3.4 ANTLR - Nástroj pro generování překladače

ANTLR nebo také ANother Tool for Language Recognition (jiný nástroj pro rozpoznávání jazyka) [10] je výkonný nástroj pro generování syntaktických analyzátorů, tzv. "parserů". Tento parser je poté schopen číst, zpracovávat, spouštět nebo překládat strukturované textové či binární soubory.

Používá se především k vytváření nových jazyků, nástrojů nebo "frameworků". Využívá bezkontextový jazyk typu LL, což je syntaktický analyzátor typu "shora-dolů" pro bezkontextové gramatiky. Analyzuje vstup zleva (Left) doprava a konstruuje nejlevější derivaci (Leftmost) věty. Gramatiky, které jsou takto analyzovatelné, se nazývají LL gramatiky.

Pro vygenerování Monkey C parseru je tedy potřeba nainstalovat ANTLRv4 (ANTLR verze 4), který je možné získat na oficiálním webu [10]. Stačí tedy stáhnout aktuální ANTLR jar, což je momentálně "antlr-4.8-complete.jar". Soubor je zakončen příponou **.jar**, z toho vyplývá, že je ANTLR napsán jazyce Java. K úspěšnému spuštění ANTLR nástroje je vyžadována verze Javy 1.6 a vyšší.

3.4.1 TestRig

ANTLR poskytuje flexibilní testovací nástroj umístěný v runtime knihovně s názvem TestRig. TestRig dokáže poskytnout spoustu informací o tom, jak "recognizéry" (parser a lexer) zpracovávají InputStream ze vstupního souboru. TestRig je spouštěn z příkazového řádku pomocí aliasu *grun*. Nabízí spoustu možností, jak zobrazit vygenerovaný syntaktický strom:

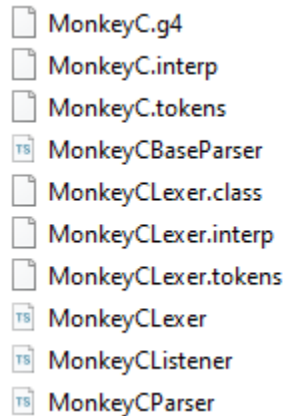
1. **-tree**: zobrazí syntaktický strom jako soubor do sebe zanořených pravidel gramatiky
2. **-tokens**: výstupem jsou tokeny, které parser vygeneroval ze vstupních dat
3. **-gui**: zobrazí syntaktický strom vizuálně v programu ParseTreeInspector, který je součástí ANTLR. Příklad zobrazení jednoduchého vstupu bez detekovaných chyb lze vidět na obrázku

```
grammar MonkeyC;

options {
    superClass=MonkeyCBaseParser;
}
@header {import { MonkeyCBaseParser } from "../MonkeyCBaseParser";}

DOT : '.';
SEMI : ';';
QUES : '?';
COLON : ':';
CLASS : 'class';
FUNCTION : 'function';
RETURN : 'return';
NEW : 'new';
VAR : 'var';
CONST : 'const';
MODULE : 'module';
USING : 'using';
AS : 'as';
ENUM : 'enum';
EXTENDS : 'extends';
```

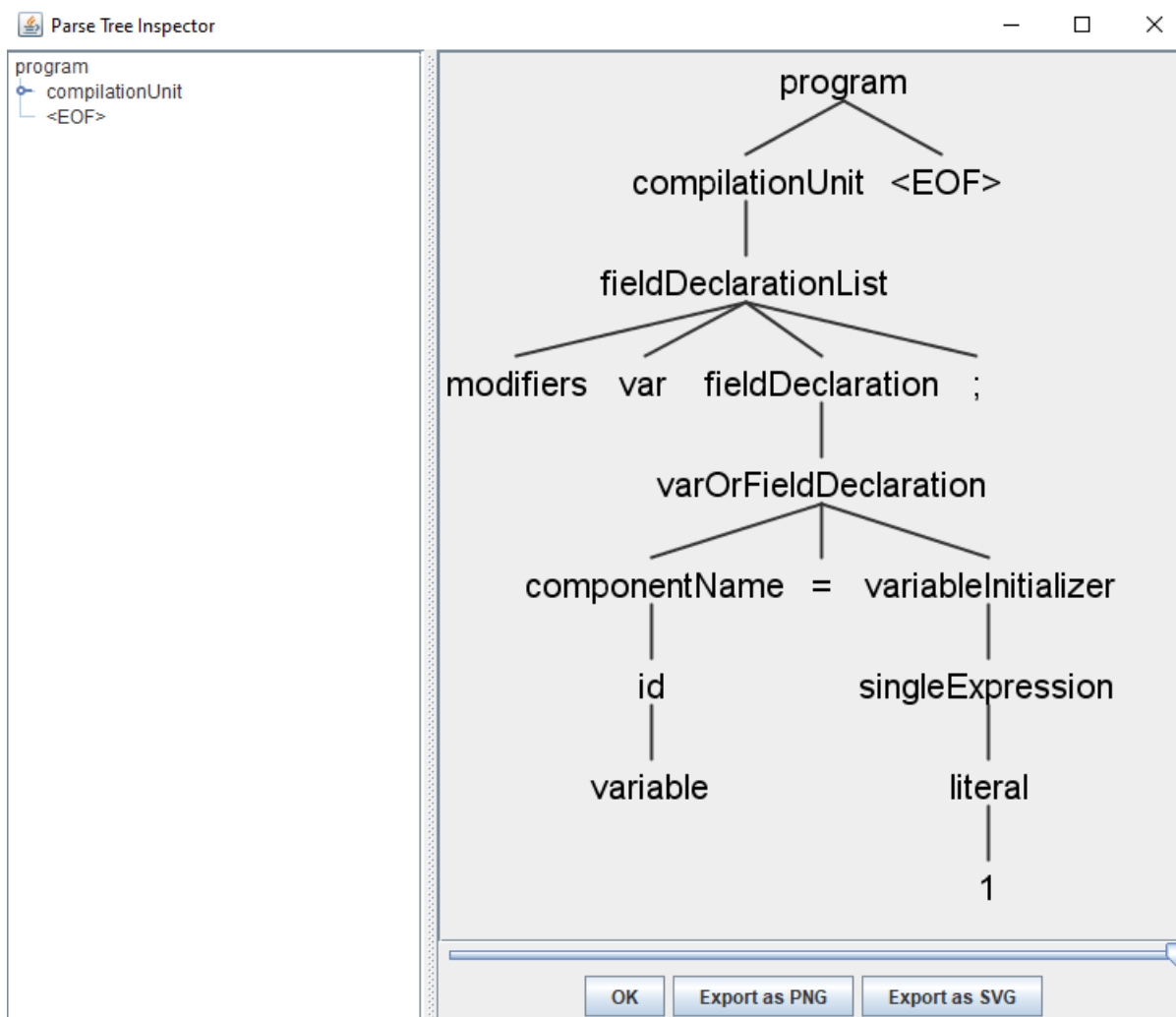
Obrázek 3.1: ukázka hlavičky gramatiky MonkeyC.g4 pro popis jazyka



Obrázek 3.2: potřebné soubory vygenerované nástrojem ANTLR

3.3. Dále lze na obrázku vidět, jak jsou mezi sebou jednotlivé části stromu, tedy uzly, navzájem propojeny. Strom začíná kořenem, jenž je v gramatice pojmenován, jako "program". Takto je kořen pojmenován při každém parsování.

Podle mého názoru se jedná o užitečný nástroj, který uživateli poskytuje přehled o tom, zda parser rozpoznal vstupní data správně, případně kde parser detekovat rozpory z gramatikou. Při vývoji a testování rozšíření byl tento nástroj velmi kvalitním pomocníkem. Možností, které pro testování gramatik a parserů TestRig nabízí, je samozřejmě více, ale pro účely této práce byly použity hlavně tři výše uvedené.



Obrázek 3.3: "ParseTreeInspector"- vizuální podoba syntaktického stromu

Kapitola 4

Syntaktická a sémantická analýza kódu

Abychom mohli implementovat náš jazyk (MonkeyC), je potřeba vytvořit aplikaci, která je schopna číst "věty" na vstupu a odpovídajícím způsobem rozpoznává klíčová slova či fráze naší gramatiky. (Obecně je jazyk složením platných vět, kdy věta se skládá z frází a fráze se skládá ze symbolů slovní zásoby).

Obecně lze říci, pokud nějaká aplikace vykonává (interpretuje) zápis jiného programu v jeho zdrojovém kódu ve zvoleném programovacím jazyce (v našem případě Typescript), že se jedná o tzv. interpret [11]. Jako příklady je možné uvést kalkulačku, aplikace pro čtení konfiguračních souborů, Python interprety, atd... Pokud, na druhou stranu, dochází k převodu "věty" z jednoho jazyka do druhého (např. z Javy do 'C Sharp'), nazývá se taková aplikace překladačem.

Úkolem překladače či interpreta je tedy rozpoznat platné vstupy, tedy věty, fráze, subfráze, klíčová slova, atd... Přičemž rozpoznáním platného vstupu je myšleno identifikovat jednotlivé fráze a rozlišit je od jiných.

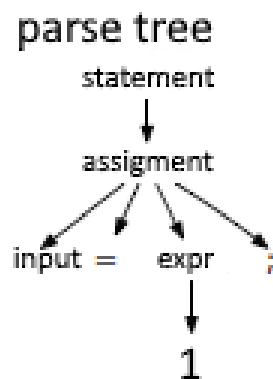
Jako příklad použijeme rozpoznání vstupu `input = 1`; s validní MonkeyC syntaxí. Syntaktický strom z těchto vstupních dat je možné vidět na obrázku 4.1. Z tohoto vstupu je patrné, že "input" je cíl přiřazení a "1" představuje hodnotu, která se má přiřadit. Stejně jako jsme my schopni rozlišit v našem jazyce sloveso od podstatného jména, je naše aplikace schopna rozlišit toto přiřazení od např. importování knihovny pomocí klíčového slova "using".

Programy, které jsou schopny rozpoznávat konkrétní jazyk, se nazývají parsery nebo syntaktické analyzátory, přičemž syntax odkazuje na pravidla obsažená v popisu jazyka.

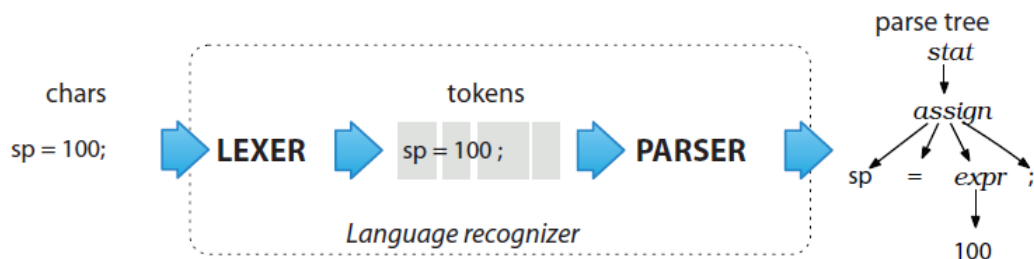
4.1 Parser a Lexer

K vytvoření parseru a lexeru je potřeba spustit ANLTR nástroj, který na základě popisu jazyka (gramatiky) tyto soubory vygeneruje. S jejich pomocí je poté možné generovat syntaktické stromy pro vstupní data. Zde platí, že pokud jsou vstupní data validní (nejsou v rozporu s pravidly gra-

matiky), tak je vygenerovaný parser vždy rozpozná správně neohledě na složitost gramatiky. Proces, při kterém dochází ke skládání znaků ze vstupních dat do slov či symbolů (tokenů) nese název **lexikální analýza** nebo **tokenizace**. Program, který provádí tuto tokenizaci se nazývá právě **lexer**. Obecně lze říci, že lexer (nebo také tokenizér) "rozdělí" text na vstupu (Monkey C kód) na tokeny. [12]. Dále lexer dokáže vytvořené tokeny seskupovat do tokenových tříd, nebo typů, např. IDENTIFIER (identifikátor), INTLITERAL (celočíslná proměnná), DOUBLELITERAL (proměnná typu double), atd... V momentě, kdy lexer rozdělil vstupní data na jednotlivé tokeny, jsou tyto tokeny předány parseru. Ten poté "shora dolů" prochází text na vstupu a porovnává jednotlivé řádky s pravidly obsažené v gramatice [12]. Na obrázku 4.2 je možné vidět, jak parser zpracovává vstup `sp = 100;`. Tokeny, které lexer z těchto dat rozpozná, jsou [SP, =, 100, ;, <EOF>], kde SP je název proměnné, 100 hodnota, která je proměnné přiřazena, jako ve většině programovacích jazyků musí být příkaz ukončen středníkem, <EOF>, který se vyskytuje na konci každého validního vstupu, detekuje konec souboru (EOF - End Of File).



Obrázek 4.1: přiřazení hodnoty 1 proměnné input



Obrázek 4.2: Ukázka, jak Language recognizer zpracovává vstupní sekvenci znaků [13]

4.2 Syntaktický strom

Syntaktický strom představuje datovou strukturu, kterou ANTLR vytvoří při parsování vstupního souboru. Struktura je složená z kořene (root) a uzlů, které mohou představovat buď další podstromy, které odpovídají pravidlům gramatiky, nebo listy stromu.

V naší aplikaci Syntaktický strom představuje třída AST (viz. kód níže), která uchovává všechna důležitá data pro korektní analýzu vstupního souboru, např. počet uzlů stromu, soubor vztahující se ke konkrétnímu stromu, kořen stromu, atd...

```
export class AST {

    static nodeCount: number = 0;
    private parseTree : Node[];
    public documentName: string;
    public currentNode : Node;
    public root : Node;

    constructor(documentName: string) {

        this.parseTree = [];
        this.documentName = documentName;
        this.currentNode = new Node(undefined,undefined,undefined,undefined,
            undefined);
        this.root= new Node(undefined,undefined,undefined,undefined,undefined);

    }

    getParseTree() {
        return this.parseTree;
    }

    addNode(node : Node) : void {
        this.parseTree.push(node);
    }

    findNode(ruleNumber: number) : Node | undefined {

        for(let i = this.parseTree.length-1; i >= 0; i--) {
```

```

        if(this.parseTree[i] != null && this.parseTree[i].getContext()?.
            ruleIndex === ruleNumber) {
            return this.parseTree[i];
        }
    }
    return undefined;
}
}

```

Listing 4.1: třída AST v jazyce Typescript

Další, neméně důležitou, třídou v naší aplikaci je třída Node, která představuje uzel stromu. Jedná se, ve své podstatě, o strukturu uchovávající následující atributy:

1. kontext daného uzlu
2. předka uzlu
3. potomka uzlu
4. hodnotu představující text v daném uzlu
5. typ uzlu, který se odvíjí od pravidel v gramatice

```

export class Node {

    private id : number;
    private context : ParserRuleContext | undefined ;
    private parent : Node | undefined;
    private child : Node[] | undefined;
    private value: string | undefined;
    private _type : number | undefined;

    constructor(context: ParserRuleContext | undefined, parent : Node | undefined,
        child : Node[] | undefined, value : string | undefined, _type: number |
        undefined)

    {
        this.id = AST.nodeCount;
        this.context = context;
        this.parent = parent;
    }
}

```

```

    this.child = child;
    this.value = value;
    this._type = _type;

    AST.nodeCount++;

}

getId() { return this.id; }
getContext() { return this.context; }
getParent() { return this.parent; }
getChildren() { return this.child; }
getValue() {return this.value; }
getType() {return this._type; }
addChild(child : Node) : void{ this.child?.push(child); }

}

```

Listing 4.2: třída Node v jazyce Typescript

4.3 ANTLR Listener a callback funkce

ANTLR disponuje dvěma mechanismy, které umožňují průchod stromem (tzv. "tree-walking mechanisms"). Jedná se o mechanismy **listener** a **visitor**, přičemž výchozím je listener. Největší rozdíl mezi nimi spočívá v tom, že listener metody jsou volány nezávisle ANTLR objektem, zatímco visitor metody vyvolávají rovněž metody potomků uzlu, na kterém se právě nachází, což způsobuje, že některé podstomy nebudou při průchodu vůbec navštíveny. Samotné listenery jsou ekvivalentní SAX objektům, které se používají v XML parserech.

Aby bylo možné stromem procházet a při průchodu vyvolat odpovídající události, ANTLR disponuje třídou `ParseTreeWalker`. Právě ona se stará o to, aby byly volány callback funkce. Další důležitou komponentou, vygenerovanou ANTLR nástrojem je rozhraní `ParseTreeListener` (v našem případě `MonkeyCListener` 3.2). Toto rozhraní definuje veškeré metody potřebné k kompletnímu průchodu stromem, např. `"enterFunctionDeclaration(context: FunctionDeclarationContext)"` či `"exitFormalParameterDeclarations(context: FormalParameterDeclarationsContext)"` 4.3. Ke každému pravidlu obsaženému v gramatice, parser vygeneruje 2 metody a kontextovou třídu. První metoda vždy začíná prefixem "enter" a druhá prefixem "exit", kontext poté odpovídá příslušnému pravidlu. Ve chvíli kdy `ParseTreeWalker` narazí na příslušný uzel, vyvolá odpovídající metodu, podle toho zda do uzlu vstupuje, nebo jej opouští.

```
enterFunctionDeclaration(context: FunctionDeclarationContext);  
exitFunctionDeclaration(context: FunctionDeclarationContext);  
enterFormalParameterDeclarations(context: FormalParameterDeclarationsContext);  
exitFormalParameterDeclarations(context: FormalParameterDeclarationsContext);
```

Obrázek 4.3: funkce vygenerované nástrojem ANTLR

Kapitola 5

Návrh a implementace rozšíření

Rozšíření bude, jak již bylo nastíněno výše, implementováno v jazyce Typescript. K vytvoření rozšíření samotného je potřeba Node.js a Git. Poté je vyžadována instalace programů "Yeoman" a "VS Code Extension Generator", pomocí kterých jsou rozšíření generovány.

5.1 Error Listener

Třída, která slouží k detekování zachytávání syntaktických chyb v kódu, čili chyb, které jsou v rozporu z pravidly gramatiky. Každá chyba, kterou rozšíření detekuje, je uložena do struktury `ErrorDescription` 5.1. Všechny chyby jsou poté uloženy v poli a pomocí metody `getSyntaxErrors()` je možné je získat. Rozšíření chyby vypisuje do konzole, jak je ve VS Code zvykem. Součástí zprávy jsou:

1. řádek, na kterém se chyba nachází
2. pozice na řádku
3. popis chyby

```
export interface ErrorDescription {  
    document: string;  
    offendingSymbol: any;  
    line: number;  
    charPositionInLine: number;  
    msg : string  
    e: RecognitionException | undefined  
}
```


5.2 Modul Toybox

Komponenta označována, jako modul, představuje v Monkey C jmenný prostor (namespace), pod kterým lze seskupovat třídy, metody, funkce, atd... Tento modul tedy obsahuje všechny potřebné třídy, které Monkey C poskytuje, na jednom místě. Z názvů jednotlivých modulů a tříd lze jednoduše odvodit, jaké metody se zde nachází a k jakému účelu slouží.

Jelikož tento modul, není nikde oficiálně dostupný. Respektive neexistuje žádná oficiální verze, kterou by bylo možné použít, bylo nutné sestavit modul vlastními silami. K jeho vytvoření bylo použito oficiální Connect IQ SDK. V tomto SDK jsou obsaženy informace, ke všem komponentům Toyboxu. Problémem bylo, že zdrojem těchto informací byly soubory ve formátu .html, tedy webové stránky. Bylo tedy nutné veškeré informace extrahovat z html elementů a následně je sestavit do požadované formy.

5.3 Zvýraznění kódu

Zvýraznění, nebo-li obarvení klíčových slov kódu je realizováno pomocí JSON souboru, jenž pomocí regulárních výrazů v textu vyhledává příslušná slova a ty poté obarvuje.

Hned na první pohled je zřejmé, že obarvení poskytuje uživateli větší přehled a orientaci v kódu, jak je vidět na obrázku 5.2

5.4 Automatické doplňování a našeptávání kódu

Jako první bylo v rozšíření řešeno našeptávání klíčových slov jazyka, např. NEW, VAR, FUNCTION, atd... k tomuto účelu byl použit "The ANTLR4 Code Completion Core"[14]. Jedná se o "stroj"sloužící k dokončování kódu pro analyzátory založené na ANTLR4. Engine c3 je schopen poskytnout kandidáty na doplnění kódu, kteří jsou užiteční pro editory s analyzátory generovanými ANTLR, nezávisle na skutečném jazyku / gramatice použité pro generování. Původní implementace je poskytována jako "node module" a je psána v jazyce Typescript, což bylo vhodné použít vzhledem k tomu, že rozšíření je také psáno v Typescriptu.

Dále bylo řešeno našeptávání lokálních funkcí a proměnných. V Monkey C jsou k tomuto účelu použity 2 prefixy **self.** a **me..** Pokud uživatel v kódu zadá jeden z těchto prefixů, rozšíření spustí funkci **provideAutocomplete**, která pomocí průchodu syntaktickým stromem všechny dostupné proměnné, funkce, či třídy, podle toho, kde se zrovna uživatel v kódu nachází.

```

1  using Toybox.WatchUi;
2  using Toybox.System;
3
4  class A {
5
6      private var privateVal;
7
8      // Update the view
9      function onUpdate(dc) {
10
11          // Get and show the current time
12          var clockTime = System.getClockTime();
13
14      }
15  }
16

```

Obrázek 5.1: ukázka neobarveného kódu v MonkeyC

```

1  using Toybox.WatchUi;
2  using Toybox.System;
3
4  class A {
5
6      private var privateVal;
7
8      // Update the view
9      function onUpdate(dc) {
10
11          // Get and show the current time
12          var clockTime = System.getClockTime();
13
14      }
15  }

```

Obrázek 5.2: ukázka obarvení kódu v MonkeyC

V neposlední řadě bylo potřeba vyřešit, jak zjistit, co se nachází v konkrétní proměnné, a na základě toho poté provést příslušné našeptávání. Toto je řešeno pomocí komentářů. Tyto Komentáře se objevují jednak v Toyboxu, kde slouží pro orientaci v už tak rozsáhlém modulu a popisu jednotlivých jeho součástí. Součástí popisu jsou informace o datových typech, vstupních parametrech (pokud se jedná o funkci), návratových typech atd...

Komentáře má také k dispozici uživatel přímo v kódu. Každá proměnná, která je deklarována, by měla na sebou obsahovat komentář nesoucí informaci o datovém typu této proměnné. Komentář má jednoduchou syntax, viz. `[src:comment]`, a rozšíření je navíc schopné jeho strukturu automaticky doplnit po zadání `/**`. Je zde využito toho, že každá proměnná v Monkey C musí být deklarována předtím, než ji lze použít. A právě na základě tohoto byly v rozšíření komentáře implementovány. Není tedy potřeba složitě hledat a ukládat informace o datovém typu do nějaké struktury, stačí pouze v kanálu komentářů najít příslušný řádek, na kterém je proměnná deklarována a z něj datový typ extrahovat.

```
/**  
 * @type Toybox.Lang.Number  
 */
```

Listing 5.2: rozhraní pro popis chyby

5.5 Nedostatky rozšíření

Při implementaci automatického našeptávání byl detekován problém, kvůli kterému není možné provést volání více funkcí po sobě na jednom řádku. Uvedme si příklad, kdy máme proměnnou typu **String**, v níž je uloženo číslo. Hodnotu v této proměnné budeme chtít převést na typ Integer, čili zavolat metodu `toNumber()`, a poté bezprostředně po volání `toNumber()` zavolat další metodu. Zde nastává problém, kdy rozšíření, jako další vstup neočekává možné volání funkce 5.6. Jádrem tohoto problému je, že poskytnutá bezkontextová gramatika popisující jazyk není stoprocentně přesná. A právě kvůli těmto "nepřesnostem" je možné při implementaci narazit na podobné komplikace. V průběhu vývoje zatím nebyly registrovány další problémy způsobené gramatikou.

Kapitola 6

Testování výsledného řešení

Testování probíhalo napříč několika fragmenty kódu. Zdrojem těchto fragmentů bylo, stejně jako pro sestavení Toyboxu, oficiální Connect IQ SKD.

Kapitola 7

Závěr

Cílem této bakalářské práce bylo nastudovat jazyk Monkey C, nástroj ANTLR, díky kterému jsme schopni analyzovat a parsovat formální jazyky, a s jeho pomocí vytvořit parser jazyka Monkey C, jenž byl následně použit k analyzování kódu.

V úvodní části této práce se hovoří o tom, jak znatelně malé zastoupení má jazyk Monkey C na trhu s rozšířeními pro VS Code, vzhledem k ostatním programovacím jazykům. Tato skutečnost byla také jeden z hlavních důvodů, které vedly k vytvoření vlastního řešení. Dále byl popsán jazyk Monkey C, který je určen k vývoji aplikací a rozšíření pro zařízení Garmin, a druhy aplikací, které je možné v Monkey C vyvíjet. Při vývoji rozšíření a následném testování jsem došel k závěru, že jazyk Monkey C má s ostatními, jako Python či Java mnoho společného.

Hlavním výsledkem práce je rozšíření pro VS Code, které uživateli napomáhá s psaním Monkey C kódu. Během vývoje byly řešeny problémy, jako jsou detekce modulů, funkcí, tříd a proměnných. Bylo řešeno našeptávání proměnných a funkcí, které jsou obsaženy ve třídě, pomocí klíčových slov **self.** a **me.**, které Monkey C využívá. Bylo řešeno parsování více souborů, pokud se jich v pracovní složce nachází více. Dále byla řešena viditelnost mezi soubory, obarvení kódu pro jeho zpřehlednění, výpis chyb prostřednictvím ErrorListeneru, detekování datového typu proměnné pomocí komentáře, jenž je její součástí, atd... Součástí práce je i vygenerovaný modul Toybox, obsahující všechny potřebné třídy a funkce. Rozšíření bylo testováno na ukázkových kódech z oficiálního Connect IQ SDK.

V této práci jsem uplatnil mnoho znalostí napříč různými jazyky. Využil jsem znalost jazyka JavaScript, jenž má prakticky totožnou syntax s Typescriptem. Dále jsem využil znalosti objektově orientovaného programování, které byly velmi užitečné při vývoji všech podpůrných tříd použitých ve finální aplikaci.

Díky této práci jsem měl také možnost rozšířit své znalosti o práci s jazykem Typescript. Dále jsem měl možnost pracovat s nástrojem ANTLR, který mě umožnil nahlédnout do problematiky překladačů a vývoji jazykových procesorů. Po dobu tvorby práce byl veškerý postup zaznamenáván pomocí verzovacího nástroje Git.

Do budoucna by bylo možné třídu `DocumentHandler` rozšířit o tabulku symbolů, která by více zefektivnila sémantickou analýzu kódu, a tím pádem zvýšila účinnost rozšíření. Dále by bylo vhodné nalézt efektivnější řešení pro generování Toybox modulu, jenž byl pro tuto práci vygenerován z html elementů obsažených v SDK. Jedním z možných řešení by bylo, kdyby společnost Garmin Ltd. [4] tento modul v nějaké ucelené podobě zveřejnila.

Literatura

1. *Essential tools for software developers and teams*. JetBrains s.r.o., [b.r.]. Dostupné také z: <https://www.jetbrains.com/>.
2. *Visual Studio Marketplace*. Microsoft Corporation, [b.r.]. Dostupné také z: <https://marketplace.visualstudio.com/vscode>.
3. *Hello Monkey C!* Garmin LTD or Its Subsidiaries, 2021. Dostupné také z: <https://developer.garmin.com/connect-iq/monkey-c/>.
4. *Garmin International: Home*. Garmin Ltd., [b.r.]. Dostupné také z: <https://www.garmin.com/en-US/>.
5. *Garmin Connect IQ: An in-depth introduction to the platform you can now use today*. 2015-01. Dostupné také z: <https://www.dcrainmaker.com/2015/01/connect-iq-intro.html>.
6. VĚNSEK, Tomáš. *Konfigurovatelná aplikace hodinek pro platformu Garmin Connect IQ*. 2019.
7. ABELSON, Harold; SUSSMAN, Gerald Jay. *Structure and interpretation of computer programs*. The MIT Press, 1996.
8. *Visual Studio Code*. Wikimedia Foundation, 2020-06. Dostupné také z: https://cs.wikipedia.org/wiki/Visual_Studio_Code.
9. *TypeScript*. Wikimedia Foundation, 2020-08. Dostupné také z: <https://cs.wikipedia.org/wiki/TypeScript>.
10. PARR, Terence. 2021. Dostupné také z: <https://www.antlr.org/>.
11. *Interpret (software)*. Wikimedia Foundation, 2020-06. Dostupné také z: [https://cs.wikipedia.org/wiki/Interpret_\(software\)](https://cs.wikipedia.org/wiki/Interpret_(software)).
12. PARR, Terence. In: *Definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013, s. 20.
13. PARR, Terence. In: *Definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013, s. 10.
14. MIKE-LISCHKE. *mike-lischke/antlr4-c3*. [B.r.]. Dostupné také z: <https://github.com/mike-lischke/antlr4-c3>.