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

Projekt 5

Vytvoření knihoven s definovaným rozhraním v jazycích C, C++, C# a FreePascal

Plzeň 2016 Petr Kozler

1 Úvod

Cílem této práce je usnadnit překlad programového kódu z programovacího jazyka *Java* do jiných programovacích jazyků. Konkrétně se bude jednat o jazyky *C*, *C++*, *C#* a *Pascal* v dialektu překladače Free Pascal. Práce bude sestávat ze čtyř hlavních kroků, z nichž první představuje část analytickou, zbylé tři pak představují část realizační.

Prvním krokem bude seznámení se s dostupností funkcionality různých základních metod z jazyka Java v ostatních výše uvedených jazycích a průzkum toho, jak v těchto jazycích fungují. Předmětem zkoumání bude především paradigma jednotlivých jazyků, dále typový systém, funkce, popř. metody pro matematické výpočty, kontejnery (zejména pole a seznamy, kdy hlavním předmětem pozornosti budou seznamy implementované dynamicky zvětšovaným polem a spojové seznamy), řetězce, funkce nebo metody pro práci s kontejnery (např. řazení a vyhledávání prvků) a funkce, popř. třídy pro vstupní a výstupní operace. Účelem zkoumání bude nalezení nejsnadnějších (s ohledem na efektivitu a přehlednost zdrojového kódu) dostupných způsobů, jak v daných jazycích vytvořit konstrukci funkčně odpovídající danému řešení z jazyka Java.

Druhým krokem bude vytvořit v těchto jazycích knihovny s rozhraním ve stylu jazyka Java, obalující tyto konstrukce. To bude realizováno způsobem, který umožní jejich použití s minimálním počtem úprav zdrojového kódu z jazyka Java, který bude tyto konstrukce využívat, se současným úplným zachováním požadované funkčnosti.

Třetím krokem bude vytvoření aplikace v jazyce Java, která bude sloužit ke správě vytvořených knihoven. Aplikace bude mít grafické uživatelské rozhraní pro snadné ovládání a bude umožňovat vyhledávání, zobrazení a popř. přidávání, úpravy a odstraňování jednotlivých knihoven i programovacích jazyků. Data (zdrojové kódy knihoven) budou ukládána do souborů ve formátu XML, která bude umožňovat snadné úpravy i bez pomoci výše uvedené aplikace (s využitím jiného editoru). Na chybné vstupy bude aplikace reagovat hlášením, z něhož bude patrné, kde se vyskytl problém, a případně nabídne vhodné možnosti řešení.

Posledním krokem bude otestování celého programového vybavení. Testování bude zahrnovat jednotkové testy, které budou mít za úkol prověřovat, zda jednotlivé funkce nebo metody implementované v jazycích C, C++, C# a Pascal zachovávají funkcionalitu odpovídajících metod v jazyce Java, a zda chování metod ve zdrojovém kódu aplikace pro správu knihoven odpovídají očekáváním. Dále bude testování zahrnovat funkční testy, které ověří funkčnost vytvořených knihoven při reálném použití ve složitějším programu.

2 Analýza metod a funkcí programovacích jazyků

V této kapitole budou popsány charakteristiky jednotlivých jazyků a různé základní funkce a metody, které tyto jazyky poskytují.

2.1 Obecná charakteristika

Základní rozdíly mezi popisovanými jazyky jsou mj. ve způsobu zpracování zdrojového kódu, programovacím paradigmatu, správě paměti a syntaxi.

2.1.1 Java

Java je názvem technologie, jejíž součástí je kromě stejnojmenného vysokoúrovňového (poskytujícího vysokou úroveň abstrakce) *objektově orientovaného* programovacího jazyka i softwarová platforma pro běh programů v tomto jazyce vytvořených. Z hlediska způsobu překladu a spuštění zdrojového kódu se nejedná o čistě kompilovaný ani interpretovaný jazyk, ale o hybridní jazyk. Zpracování programového kódu probíhá následovně [78, 79, 96, 131]:

1. Nejprve je proveden překlad veškerého zdrojového kódu (ukládaného do textových souborů, které mají příponu `.java`) s pomocí kompilátoru `javac` tzv. mezijazyka zvaného *bajtkód*, angl. bytecode (ukládaného v souborech s příponou `.class`). Bytecode je jednodušší formou kódu - jedná se o strojový kód *virtuálního stroje* zvaného Java Virtual Machine (dále jen JVM).

2. Virtuální stroj JVM následně interpretuje (po spuštění nástrojem `java`) kód `.class` souboru do nativního kódu cílového procesoru.

Stejný kód je tedy možné spustit na různých platformách (resp. na platformách, pro které je JVM k dispozici) [131].

Jednou ze základních vlastností jazyka Java je *automatická správa paměti*. Ta umožňuje vytváření objektů bez nutnosti jejich explicitního odstranění za účelem uvolnění paměti ve chvíli, kdy již nejsou potřeba. Objekty, které se nadále nepoužívají (nevedou již na ně žádné odkazy), jsou běhovým prostředím odstraňovány automaticky. Proces automatického likvidování objektů se nazývá garbage collection [96, 131]. Syntax jazyka je zjednodušenou a mírně upravenou podobou syntaxe jazyků C a C++. Objektový model jazyka je

založený na třídách a jazyk podporuje genericitu. S výjimkou základních datových typů jsou všechny proměnné objektem (instancí třídy). Kořenem stromové hierarchie dědičnosti tříd je třída `java.lang.Object` – každá třída je tedy potomkem třídy `java.lang.Object`. Třídy, popř. další konstrukce (rozhraní, výčtové typy) mohou být seskupovány do jmenných prostorů, tzv. balíků (packages) [84, 131].

2.1.2 C

C je nízkoúrovňový, strukturovaný, na funkcích založený jazyk. Pojem nízkoúrovňový znamená, že jazyk přímo pracuje jen se základními datovými typy (znaky, čísla), poskytuje tedy nižší úroveň abstrakce než objektově orientované jazyky jako je např. jazyk Java. Výhodou tohoto přístupu je relativní jednoduchost vytvoření kompilátoru pro konkrétní platformu a efektivita kódu [1, 128].

Jak již bylo nastíněno, na rozdíl od jazyka Java se jedná o jazyk čistě kompilovaný. Kompilace zdrojového kódu jazyka C sestává z několika fází [1, 128]:

1. *Předzpracování* - neboli zpracování kódu tzv. preprocesorem, které spočívá v úpravě kódu do podoby pro snadnější překlad. Součástí této úpravy je např. odstranění komentářů, rozvoj maker, vložení tzv. hlavičkových souborů (soubory s příponu `.h`, obvykle obsahující např. prototypy funkcí, deklarace globálních proměnných nebo definice konstant a globálních typů, a obvykle neobsahující výkonný kód) atd.

2. *Překlad* - program vykonávající tuto úlohu se nazývá kompilátor neboli překladač (angl. compiler). Jedná se o hlavní část zpracování zdrojového kódu. Textový soubor kódu upraveného preprocesorem je převeden do objektového neboli relativního kódu cílového počítače (výsledkem práce kompilátoru je soubor `.obj`). Objektový kód představuje téměř hotový spustitelný soubor, pouze adresy proměnných a funkcí jsou zatím zapsány relativně (nejsou zatím známé, protože jsou např. uložené v externí knihovně).

3. *Sestavení* - zpracování relativního kódu vytvořeného překladačem pomocí sestavovacího programu (angl. linker) zahrnuje přidělení absolutních adres a nalezení adres dosud neznámých identifikátorů (jako jsou volané funkce z knihoven). Výstupem této fáze je již hotový program - spustitelný soubor (v OS Windows s příponou `.exe`).

Ze syntaxe tohoto jazyka je odvozena syntaxe všech ostatních jazyků uvedených v této práci s výjimkou jazyka Pascal. Na rozdíl od jazyků Java

a C# zde neexistuje mechanismus automatické správy paměti – alokace i dealokace paměti je úkolem programátora. K tomuto účelu nicméně jazyk C poskytuje standardní knihovní funkce. Vzhledem k nutnosti *manuální správy paměti* obsahuje jazyk C na rozdíl od Javy nejen proměnné, obsahující přímo hodnoty, ale též proměnné nazývané jako ukazatele neboli směrníky (angl. pointery) na hodnoty daných typů. Ukazatel je proměnná, jejíž hodnotou je adresa v paměti a na této adrese je teprve uložena hodnota (kromě ukazatelů na proměnné existují též ukazatele na funkce). Právě s pomocí ukazatelů jsou v jazyce C realizovány konstrukce podobné složitějším typům jazyka Java, jako jsou řetězce a pole. Objekty jazyka Java lze částečně napodobit s využitím konstrukcí jazyka C pro vytváření nehomogenních (složených z různých datových typů) stavových struktur [1, 128].

2.1.3 C++

C++ je jazyk, který je nadstavbou jazyka C - každý platný program v jazyku C je tedy zároveň platným programem v jazyku C++. Princip zpracování zdrojového kódu je stejný jako v případě jazyka C (některé překladače fungují, nebo dříve fungovaly tak, že nejprve převedli zdrojový kód C++ na zdrojový kód C, který byl poté předán překladači jazyka C) [129].

Stejně jako jazyk C i jazyk C++ vyžaduje manuální správu paměti. Pro tento účel jsou nicméně k dispozici nové jazykové prostředky, např. konstruktory a destruktory nebo reference na proměnné (jako alternativa k ukazatelům). Kromě toho obsahuje jazyk C++ řadu dalších nových konstrukcí, jednou z nejdůležitějších je třída. Rozdílem oproti C je tedy mj. podpora objektově orientovaného programování (vedle procedurálního přístupu z jazyka C) a generického programování (prostřednictvím tzv. šablon). Jazyk C++ lze proto označit jako multiparadigmatický [7, 129].

Mezi další nové konstrukce patří jmenné prostory (obdoba balíků v jazyku Java), přetěžování funkcí i operátorů, standardní parametry, výše zmíněné šablony, vložené (angl. inline) funkce a další. Kromě nových konstrukcí přináší C++ oproti C také rozsáhlou sadu nových knihoven, přičemž je stále možné používat i knihovny z jazyka C [129].

2.1.4 C#

C# je vysokoúrovňový, objektově orientovaný jazyk vycházející z jazyků, jako je C++ a Microsoft Visual Basic a je zaměřený především na vývoj aplikací pro platformu .NET Framework [32, 132].

Obdobou JVM je *Common Language Runtime* a obdobou bytecode je *Microsoft Intermediate Language*. Syntax je velmi podobná jazyku Java, na-

víc ale obsahuje konstrukce jako například částečné třídy (třídy, jejichž definice je rozdělena do více zdrojových souborů), delegáty, přetěžování operátorů (nejen metod), předávání parametrů metod referencí nebo bezznaménkové číselné datové typy [36, 46, 51, 58, 132].

2.1.5 Pascal

Pascal je, stejně jako C a C++, kompilovaný jazyk, pro který je k dispozici řada překladačů (každý z nich definuje mírně odlišnou podobu jazyka, tzv. dialekt). Jedním z nich je i volně šiřitelný překladač *Free Pascal*, který je použit v rámci této práce. Všechny poznámky týkající se Pascalu se tedy vztahují na dialekt překladače Free Pascal a nemusejí platit pro ostatní překladače [130].

Vedle způsobu zpracování kódu je Pascal srovnatelný s jazykem C++ také po stránce správy paměti a v podporovaných programovacích paradigmatech - je tedy podporováno jak strukturované programování založené na procedurách a funkcích (jako procedury jsou nazývány podprogramy bez návratové hodnoty), tak objektově orientované programování [130].

Na první pohled odlišná je syntax jazyka, která je méně úsporná a je přizpůsobena pro výuku. Soubor zdrojového kódu programu se nazývá modul nebo jednotka (angl. unit), obvyklá přípona souboru je `.pas` a zdrojový kód má pevně danou strukturu. Jeho základem je blok, který se skládá z hlavičky, deklarace a příkazové části. Deklarační část může obsahovat podprogramy, které mají stejnou strukturu [97, 130].

2.2 Základní datové typy

Všechny uvedené jazyky jsou typované - při deklaraci proměnné je nutné uvést její datový typ. Každý z těchto jazyků poskytuje množinu základních datových typů. Jejich seznam spolu s uvedenými rozsahy hodnot je uveden níže.

2.2.1 Java

Jazyk Java je (stejně jako ostatní uvedené jazyky) *staticky typovaný*, což znamená, že každou proměnnou je nutné před použitím deklarovat a kromě jména je nutné uvést její datový typ, který určuje množinu povolených hodnot a operací nad nimi. K dispozici je 8 primitivních datových typů (typů předdefinovaných jazykem, jejichž název je klíčovým slovem jazyka), z toho 4 celočíselné, 2 pro reálná čísla, 1 znakový a 1 pro logické hodnoty. Všechny číselné datové typy jsou znaménkové [95, 131].

Tabulka 2.1: Základní datové typy jazyka Java

Název	**Význam**	**Rozsah**
`byte`	celé číslo	-2^7 .. $2^7 - 1$
`short`	celé číslo	-2^{15} .. $2^{15} - 1$
`int`	celé číslo	-2^{31} .. $2^{31} - 1$
`long`	celé číslo	-2^{63} .. $2^{63} - 1$
`float`	reálné číslo	dle IEEE 754 (jednoduchá přesnost)
`double`	reálné číslo	dle IEEE 754 (dvojitá přesnost)
`boolean`	logická hodnota	`true` nebo `false`
`char`	znak Unicode	`'\u0000'` (0) .. `'\uffff'` ($2^{16} - 1$)

Jednotlivé datové typy, jejich význam, velikost v paměti a rozsahy hodnot popisuje tabulka 2.1. Pro uvedené typy jsou k dispozici tzv. *obalové třídy*, umožňující další operace; např. `java.lang.Integer` nebo `java.lang.Long` umožňující použití hodnot číselných datových typů jako bezznaménkových. Kromě těchto datových typů poskytuje jazyk Java speciální podporu pro řetězce reprezentované třídou `java.lang.String` (popsány v kapitole 2.4) [65–67, 72, 73, 77, 82, 88, 95].

2.2.2 C

Primitivní datové typy jazyka C zahrnují typy celočíselné a typy pro reálná čísla (typ char používaný pro uchovávání znaků lze považovat za celočíselný typ a vestavěný typ pro logické hodnoty jazyk C neobsahuje). Velikosti a rozsahy jednotlivých datových typů závisí na cílové platformě. Garantovány jsou pouze minimální rozsahy a pořadí velikosti rozsahů pro jednotlivé datové typy (není však vyžadováno, aby jednotlivé datové typy měly rozsahy rozdílné velikosti) [1].

Minimální rozsahy jsou uvedené v tabulce 2.2. Typy jsou seřazeny vzestupně podle minimální velikosti. Kromě uvedených datových typů existuje ke každému celočíselnému typu bezznaménková varianta - její název začíná řetězcem `unsigned`. Takový typ má stejnou velikost jako příslušný znaménkový typ a jeho rozsah je posunutý tak, že nejmenší hodnota je 0 (tedy pouze kladná čísla). Celočíselné datové typy `long long int` a `unsigned long long int` nejsou součástí standardu C89. Konstanty představující hraniční hodnoty datových typů pro reálná čísla jsou uloženy v hlavičkovém souboru `float.h` a platí pro ně, že typ `float` má nejmenší rozsah a zbylé dva typy mají rozsah minimálně tak velký, ale obvykle větší, než `float`. Konstanty plnící stejný účel pro celočíselné typy definuje soubor `limits.h` [1].

Tabulka 2.2: Některé základní datové typy jazyka C

Název	**Význam**	**Rozsah**
`char`	celé číslo	-2^7 .. $2^7 - 1$
`short int`	celé číslo	-2^{15} .. $2^{15} - 1$
`int`	celé číslo	-2^{31} .. $2^{31} - 1$
`long int`	celé číslo	-2^{31} .. $2^{31} - 1$
`long long int`	celé číslo	-2^{63} .. $2^{63} - 1$
`float`	reálné číslo	`FLT_MIN` .. `FLT_MAX`
`double`	reálné číslo	`DBL_MIN` .. `DBL_MAX`
`long double`	reálné číslo	`LDBL_MIN` .. `LDBL_MAX`

2.2.3 C++

Jak již bylo uvedeno v jedné z předchozích kapitol, C++ je nadstavbou jazyka C, proto každý datový typ, který obsahuje jazyk C, obsahuje i C++, a platí pro něj stejná pravidla, jako v jazyku C. Jazyk C++ nicméně obsahuje nové fundamentální datových typy navíc, příkladem je datový typ pro logické hodnoty `bool`, který je obdobou typu `boolean` v jazyku Java [28, 129].

2.2.4 C#

Na rozdíl od jazyka Java jsou v jazyku C# všechna klíčová slova, představující jednoduché vestavěné datové typy ve skutečnosti *aliasy* pro třídy předdefinované ve jmenném prostoru System. Ty zde kromě znakového, logického a číselných datových typů (které jsou podobné těm v jazyku Java a rovněž mají garantované velikosti a rozsahy) zahrnují i řetězce (popsány v kapitole 2.4) a obecné objekty [33, 35, 41, 42, 45, 49, 53, 54, 58, 132].

Příklady předdefinovaných typů jsou popsány v tabulce 2.3. Dále ke každému číselnému datovému typu existují i bezznaménkové varianty stejné velikosti `byte`, `ushort`, `uint` a `ulong`. Pro popis reálných čísel je mimo typů uvedených v tabulce k dispozici typ `decimal`, který má menší rozsah, ale větší přesnost. Pro řetězec Unicode znaků je v C# k dispozici předdefinovaný typ `string` a pro obecný objekt typ `object` [34, 40, 52, 55, 58–61, 132].

2.2.5 Pascal

Typový systém jazyka Pascal obsahuje celou řadu základních datových typů (větší množství než ostatní uvedené jazyky), které se člení na typy základní (ty se dále dělí na ordinální, zahrnující mj. typy celočíselné a logické, a reálné), znakové a další [100, 102, 124].

Tabulka 2.3: Některé základní datové typy jazyka C#

Název	**Význam**	**Rozsah**
`sbyte`	celé číslo	-2^7 .. $2^7 - 1$
`short`	celé číslo	-2^{15} .. $2^{15} - 1$
`int`	celé číslo	-2^{31} .. $2^{31} - 1$
`long`	celé číslo	-2^{63} .. $2^{63} - 1$
`float`	reálné číslo	dle IEEE 754 (jednoduchá přesnost)
`double`	reálné číslo	dle IEEE 754 (dvojitá přesnost)
`bool`	logická hodnota	`true` nebo `false`
`char`	znak Unicode	`'\u0000'` (0) .. `'\uffff'` ($2^{16} - 1$)

Tabulka 2.4: Některé základní datové typy jazyka Pascal

Název	**Význam**	**Rozsah**
`Shortint`	celé číslo	-2^7 .. $2^7 - 1$
`Smallint`	celé číslo	-2^{15} .. $2^{15} - 1$
`Longint`	celé číslo	-2^{31} .. $2^{31} - 1$
`Int64`	celé číslo	-2^{63} .. $2^{63} - 1$
`Single`	reálné číslo	dle IEEE 754 (jednoduchá přesnost)
`Double`	reálné číslo	dle IEEE 754 (dvojitá přesnost)
`Boolean`	logická hodnota	`True` nebo `False`
`Char`	znak	`'#0'` (0) .. `'#255'` ($2^8 - 1$)

Pro účely této práce zde budou v tabulce 2.4 popsány pouze typy, které lze použít jako obdobu primitivních datových typů jazyka Java. Vynechány budou číselné a logické typy jiných rozsahů, než jaké poskytují základní typy v jazyce Java (například bezznaménkové celočíselné typy `Byte`, `Word`, `Longword` a `QWord` nebo některé typy pro reálná čísla, např. `Currency`) a stejně tak budou vynechány typy, jejichž velikost je platformově závislá a pro použití jako obdoby typů jazyka Java jsou tedy méně vhodné, než jejich alternativy s pevnou velikostí (zejména `Integer` a `Real`) [101, 104, 109, 124].

Typ `Char` zde slouží jako synonymum pro typ `AnsiChar`, který má velikost 1 byte. Vedle něho existuje v jazyku Pascal i typ `WideChar` s velikostí 2 byte, který reprezentuje Unicode znak [98, 124, 125].

Problematika znaků a řetězců bude podrobněji rozebrána v kapitole 2.4.

2.3 Pole a seznamy

Pro uchovávání většího množství hodnot bez nutnosti vytváření proměnné pro každou proměnnou zvlášť poskytují uvedené programovací jazyky datovou strukturu pole, případně složitější datové struktury, jako jsou dynamicky alokované seznamy.

2.3.1 Java

Pole v jazyku Java je objekt, který uchovává předem určené množství hodnot jednoho (libovolného) typu. Délka daného pole (počet prvků) je určena v průběhu jeho vytvoření a dále se nemění. Číslování prvků začíná od indexu 0 [62].

Pro pokročilejší manipulaci s poli je k dispozici knihovní třída `java.util.Arrays`. Tato třída poskytuje metody pro kopírování, porovnávání, naplnění či řazení polí hodnot různých typů, jejich převod do řetězcové reprezentaci nebo binární vyhledávání prvků [64].

Jazyk Java dále poskytuje třídy implementující pokročilejší dynamické datové struktury pro uchovávání prvků (také nazývané jako kolekce). Tyto třídy implementují rozhraní `java.util.Collection<E>` a zahrnují např. polem implementovaný seznam (třída `java.util.ArrayList<E>`), spojový seznam (třída `java.util.LinkedList<E>`) nebo množiny (např. `java.util.HashSet<E>`). Mezi další datové struktury, které jsou k dispozici, patří např. mapy (`java.util.HashMap<K, V>`) [63, 68, 74, 75, 80].

Pro účely této práce jsou důležité především seznamy. Třídy `java.util.ArrayList<E>` a `java.util.LinkedList<E>` implementují rozhraní `java.util.List<E>`, mezi jehož metody patří metody pro vkládání, úpravu, odstraňování a výběr prvků, zjišťování aktuálního počtu prvků nebo převod prvků seznamu do textové reprezentace [63, 80, 81].

Další manipulaci se seznamy (případně jinými kolekcemi) umožňuje třída `java.util.Collections`, která je obdobou třídy `java.util.Arrays`. Mezi poskytované metody tak patří kopírování, řazení, vyhledávání a další [69].

Seznamy `java.util.ArrayList<E>` a `java.util.LinkedList<E>` jsou generické, stejně jako další kolekce, metody třídy `java.util.Collections` a některé metody třídy `java.util.Arrays` [63, 64, 69, 80].

Řazení polí a seznamů, vyhledávání jejich prvků a porovnávání prvků mezi sebou se bude podrobněji věnovat kapitola 2.5.

2.3.2 C

V jazyku C je pole definováno jako posloupnost prvků stejného typu uložených za sebou v paměti. Velikost pole je určena při deklaraci. Stejně jako v jazyku Java se zde indexy prvků číslují od 0 [3].

Pole zde úzce souvisí s ukazateli. Ukazatel uchovává adresu v paměti pro danou proměnnou. Při uložení více prvků stejné hodnoty za sebou v paměti je hodnota na adrese, kam směřuje ukazatel, totéž jako první prvek odpovídajícího pole; přičtení určitého čísla k adrese ukazatele (s využitím ukazatelové aritmetiky) je pak totéž jako přistoupení k prvku pole na odpovídajícím indexu. Pole je vždy možné zkonvertovat na ukazatel příslušného typu [16].

Pro realizaci dynamicky zvětšovaných datových struktur lze použít knihovní funkce pro alokaci paměti za běhu programu. Příkladem jsou funkce `malloc` a `calloc` definované v hlavičkovém souboru `stdlib.h`, které vrací ukazatel na začátek souvislého bloku nově alokované paměti [5, 14].

2.3.3 C++

Pro obyčejná pole zde platí stejná pravidla jako pro pole v jazyku C. Pro pokročilejší datové struktury jsou nicméně k dispozici generické knihovní třídy jako `std::vector`, `std::list`, `std::set`, `std::map` a další [13, 15, 20, 29].

Seznam vytvořený jako dynamické pole je implementován třídou `std::vector`, spojový seznam pak třídou `std::list`. Jejich členské funkce (metody) pak poskytují podobnou funkcionalitu jako jejich protějšky `ArrayList` a `LinkedList` v jazyku Java. [13, 29]

2.3.4 C#

Úlohu pole v jazyku C# plní třída `System.Array`. Tato třída zároveň poskytuje funkcionalitu srovnatelnou s třídou `Arrays` jazyka Java, tedy metody pro řazení, vyhledávání, kopírování atd. Indexy se číslují stejně jako v jazyce Java [30, 31].

Obdobu `ArrayListu` jazyka Java zde představuje generická třída `System.Collections.Generic.List<T>` a pro spojový seznam je k dispozici generická třída `System.Collections.Generic.LinkedList<T>`. Obdobně jako v případě třídy `System.Array`, i tyto třídy mají metody jak pro základní operace (přidávání prvků, výběr apod.), tak pro pokročilé operace v duchu třídy `Collections` z jazyka Java [47, 48].

2.3.5 Pascal

Pole se v jazyce Pascal řadí mezi tzv. strukturované typy (typy, které uchovávají více hodnot pomocí jedné proměnné). Pascal obsahuje jak statická (velikost, resp. počáteční a konečný index je určen při deklaraci), tak dynamická (velikost je určena za běhu programu funkcí `SetLength`) pole. U statických polí závisí číslování indexů prvků na deklaraci, u dynamických se čísluje od 0 [103, 111, 113].
Kromě statických a dynamických polí Pascal, stejně jako C, podporuje ukazatele [106].

Seznam s proměnlivou velikostí je v Pascalu podporován prostřednictvím několika tříd, jako je `TList`, `TFPList` a generický `TFPGList` a `TFPGObject-List`, s rozhraním podobným rozhraní `List` z Javy [115–117, 119].

2.4 Textové řetězce

Uvedené jazyky obsahují rozdílné datové struktury, příp. datové typy pro práci s textem. Dále poskytují různé funkce nebo metody pro základní operace nad těmito strukturami.

2.4.1 Java

Řetězce znaků reprezentují instance třídy `java.lang.String`. Ty jsou neměnné - po vytvoření řetězce nelze změnit jeho hodnotu, pouze vytvořit nový (pro většího množství úprav jednoho textu jsou k dispozici třídy jako `java.lang.StringBuffer` a `java.lang.StringBuilder`). Kromě řetězcového literálu lze objekt `java.lang.String` vytvořit např. pomocí pole bajtů nebo znaků. Řetězce lze spojovat (tzv. konkatenace) pomocí operátoru pro sčítání [89–92].

Třída `java.lang.String` má metody pro různé operace nad řetězci. Patří mezi ně metody pro porovnávání řetězců, určování délky, hledání nebo vytváření podřetězců, nahrazování znaků, převody na velká nebo malá písmena, odstraňování bílých znaků (trimování) a v neposlední řadě např. rozdělování řetězců do polí podřetězců podle určené množiny oddělovačů [89].

Metody pro získání hodnoty jiného (např. číselného) datového typu z její řetězcové reprezentace poskytují obalové třídy příslušných typů [65–67, 72, 73, 77, 82, 88, 89].

K rozdělování řetězce do podřetězců (tokenů) podle oddělovacích znaků (delimiterů) je možné použít rovněž třídu
`java.util.StringTokenizer`. Při vytváření instance je určen řetězec k rozdělení a řetězec oddělovačů (každý jednotlivý znak tohoto řetězce je považo-

ván za oddělovač). Posléze jsou postupně načítány jednotlivé tokeny (v průběhu načítání lze zjišťovat jejich počet) [93].

2.4.2 C

Pro reprezentaci řetězců se obvykle používá pole hodnot typu `char`. Řetězec v C je tedy polem znaků, které je zakončené ukončovací značkou, tzv. terminátorem neboli nulovým znakem (má hodnotu 0) [6].

Funkce pro manipulaci s řetězci definuje hlavičkový soubor `string.h`. Jedná se např. o funkce pro porovnávání (`strcmp`), kopírování (`strcpy`) a konkatenaci (`strcat`) řetězců, zjišťování délky řetězce (`strlen`), a také načítání tokenů z řetězce podle určených oddělovačů (`strtok`) v duchu třídy `StringTokenizer` z Javy [11].

Soubor `stdlib.h` pak definuje mj. funkce pro získání hodnot číselných typů z jejich řetězcové reprezentace - `strtol`, `strtod` apod [10].

Zásadní odlišností znakového typu `char` (a tím i řetězci) v jazyku Java (a C#) oproti typu char jazyka C je jeho velikost - Java používá 16-bitové *Unicode* znaky, zatímco typ `char` v C je pouze 8-bitový [28].

Pro účely této práce je nicméně možné se omezit na 8-bitové kódování. Pro jednoduchost implementace knihoven v rámci realizační části této práce tak budou v jazycích C, C++ a Pascal používány pouze základní 8-bitové znaky a odpovídající implementace řetězců.

2.4.3 C++

C++ oproti C obsahuje zlepšenou podporu řetězců prostřednictvím třídy `std::string` definované v hlavičkovém souboru `string` standardní knihovny jazyka C++. Tato třída poskytuje některé metody se stejnou funkcionalitou jako metody třídy `String` z Javy. Prostřednictvím přetížení operátoru sčítání taktéž umožňuje konkatenaci řetězcových literálů ve stylu jazyka Java. Dále umožňuje získání ekvivalentního pole znaků reprezentujícího řetězec ve stylu jazyka C nebo převod textové reprezentace číselné hodnoty na odpovídající číselnou hodnotu daného typu [26, 27].

2.4.4 C#

Třída `System.String` je obdobou stejnojmenné třídy jazyka Java. Zde je možné použít alias `string`. Vnitřně je text uložen jako posloupnost Unicode znaků. Pro objekty `string` zde platí obdobná pravidla jako pro objekty `String` v Javě, včetně funkcionality poskytovaných konstruktorů a metod.

Navíc je ale možné například přistupovat k jednotlivým znakům řetězce stejným způsobem, jako k prvkům pole [56].

K rozdělování řetězce na tokeny se zde používá instanční metoda `Split`. Tato metoda vrací pole vzniklých podřetězců rozdělených podle předaných oddělovačů. Jednotlivé oddělovače zde mohou být nejen znaky, ale i celé řetězce (jsou tedy předány jako pole řetězců, nikoliv jako jediný řetězec, jehož znaky jsou jednotlivé oddělovače) [57].

2.4.5 Pascal

Řetězce jako posloupnosti 8-bitových znaků představuje typ `AnsiString`. Pro takové řetězce existuje i typ `ShortString`, ten má ale omezenou délku a jeho použití pro účely této práce není vhodné. Vnitřně je řetězec typu `AnsiString` realizován jako ukazatel a je zakončen nulovým znakem; ten však neslouží pro určení délky řetězce, nulové znaky je tak možné použít i jako součást řetězce [99, 110, 112].

Pro manipulaci s řetězci jsou poskytnuty různé funkce, například `Length` pro určení délky, `Copy` pro kopírování, `Pos` pro nalezení znaku, dále `Trim`, `UpperCase`, `LowerCase` a řada dalších [114].

K rozdělení řetězce na tokeny je možné využít třídy `TStringList`, resp. `TStrings` [122, 123].

2.5 Řazení a vyhledávání

V uvedených programovacích jazycích jsou dostupné funkce nebo metody pro řazení prvků v polích a další operace s poli. Tyto operace vyžadují v jednotlivých jazycích rozdílné způsoby porovnávání prvků.

2.5.1 Java

Řazení a binární vyhledávání prvků v poli nebo seznamu zajišťují metody `sort`, resp. `binarySearch`, kterými disponují knihovní třídy `java.util.Arrays`, resp. `java.util.Collections` [64, 69].

Pro seřazení pole primitivních datových typů je implementován algoritmus *Quicksort*. Pro řazení pole nebo seznamu objektů je použit stabilní řadící algoritmus *Mergesort*. Prvky jsou řazeny vzestupně [64, 69].

Pro všechny řazené prvky musí platit, že implementují rozhraní `java.lang.Comparable<T>` a jsou vzájemně porovnatelné. Alternativou je řazení objektů s využitím komparátoru (třída implementující rozhraní `java.util-`

`.Comparator<T>`). Stejné pravidlo pro porovnávání pak platí v případě binárního vyhledávání [64, 69–71].

2.5.2 C

Pro účely řazení a binární vyhledávání prvků definuje hlavičkový soubor `stdlib.h` funkce `qsort` (implementující algoritmus Quicksort) a `bsearch`. Obě funkce přijímají ukazatel na obecné pole, kromě počtu jeho prvků je tak nutné dále uvést velikost jednoho prvku v bajtech a dále ukazatel na funkci pro vzájemné porovnání dvou prvků tohoto pole [4, 18].

2.5.3 C++

Hlavičkový soubor `algorithm` definuje celou řadu šablon funkcí pro řazení a vyhledávání prvků v různých posloupnostech prvků včetně polí a seznamů. Pro kompletní vzestupné řazení prvků v definovaném rozsahu je určena šablona funkce `std::sort`, resp. `std::stable_sort` v případě požadavku na stabilní řazení. Pro binární vyhledávání je určena šablona funkce `std::binary_search` [21, 24, 25].

Všechny uvedené funkce kromě iterátoru na první a poslední prvek posloupnosti přejímají též ukazatel na funkci, nebo funkční objekt (instance třídy, která má jako veřejnou instanční metodu operátor volání funkce) pro vzájemné porování dvou prvků [21, 24, 25].

2.5.4 C#

Třída `System.Array`, která reprezentuje pole, a třída `System.Collections.Generic.List<T>`, která představuje generický seznam implementovaný polem proměnlivé délky, poskytují metody mj. metody pro řazení a vyhledávání prvků [30, 48].

Stejně jako v případě jazyka Java, i zde je několik možností, jak při řazení nebo vyhledávání porovnávat objekty. Jednou z možností je implementace rozhraní `System.IComparable<T>`. Další možností je poskytnutí příslušného komparátoru - objektu, jež implementuje rozhraní `System.Collections.Generic.IComparer<T>`. Navíc je zde možné využití delegátů - namísto porovnávací třídy se v tomto případě poskytne odkaz na porovnávací metodu `System.Comparison<T>` [37, 43, 44].

2.5.5 Pascal

Pro třídy, implementující seznamy (jako `TStringList`, `TList`, `TFPList` a `TFPGList`) jsou k dispozici metody pro řazení prvků algoritmem Quicksort, v případě některých (`TStringList`) též pro binární vyhledávání [118, 120].

Pro porovnávání prvků přijímají dotyčné metody jako parametr porovnávací funkci [121].

2.6 Matematické výpočty

Každý uvedený jazyk vedle nejzákladnějších aritmetických operátorů disponuje knihovnou poskytující běžné matematické funkce např. pro určení absolutní hodnoty, nalezení menšího nebo většího prvku nebo výpočet odmocniny.

2.6.1 Java

Metody představující běžné matematické funkce jsou k dispozici v knihovní třídě `java.lang.Math`. Mezi podporované matematické funkce patří absolutní hodnota, mocniny a odmocniny, goniometrické a cyklometrické funkce, logaritmy, hledání minima a maxima z dvojice čísel nebo zaokrouhlování. Dále obsahuje definice běžných matematických konstant, čísla π a čísla e [83].

2.6.2 C

Funkce pro určení absolutní hodnoty (např. `abs`) a celočíselné dělení (např. `div`) jsou definovány v souboru `stdlib.h`. Funkce pro ostatní matematické výpočty v souboru `math.h` a zahrnují mj. většinu funkcí uvedených u třídy `Math` v jazyku Java. Příklady: funkce `pow`, `sqrt`, `sin`, `cos`, `tan`, `log`, `log10`, `min`, `max`, `round` [8, 10].

2.6.3 C++

Soubor math pro C++ definuje dodatečné knihovní funkce pro matematické výpočty, jako je například funkce `::abs` pro výpočet absolutní hodnoty přetížená pro různé celočíselné typy [2, 10].

2.6.4 C#

Obdobně jako v jazyku Java, i zde existuje knihovní třída `System.Math`, která plní stejnou úlohu. Množina dostupných matematických funkcí a konstant je obdobná jako v případě knihovny `Math` v jazyku Java [50].

2.6.5 Pascal

Některé matematické funkce (vedle celé řady jiných), jako např. `abs`, `sin`, `cos`, `tan`, `Power`, `sqr`, `sqrt` nebo `round` jsou definovány v modulu `System`, další jsou pak pokryty dodatečně v modulu `Math` (příklady: `min`, `max`, `log10`, `RoundTo`) [105, 114].

2.7 Vstup a výstup

Součástí uvedených jazyků jsou knihovny pro vstupní a výstupní operace pro standardní vstup a výstup, pro soubory, sockety atd.

2.7.1 Java

Pro načítání uživatelského vstupu je v jazyce Java k dispozici několik tříd, jednou z nich je třída `java.util.Scanner`. `java.util.Scanner` načítá vstup ze zdroje předaného v konstruktoru, kterým může být např. soubor, již existující textový řetězec nebo instance třídy `java.io.InputStream`, kterou může být např. standardní vstup (výchozí objekt odkazovaný statickým atributem `in` knihovní třídy `java.lang.System`) nebo vstupní proud (angl. stream) ze socketu [76, 87, 94].

`java.util.Scanner` obsahuje metody jak pro čtení řádek, tak čtení jednotlivých tokenů buď jako řetězců, nebo jako hodnot různých jiných datových typů, např. čísel. Tokeny načtené jako řetězce je taktéž možné dodatečně převést na hodnoty jiných typů pomocí metod s názvem začínajícím na řetězec `parse-` obalových tříd příslušných datových typů (metody `toString` těchto tříd provádějí opačnou operaci) [65–67, 72, 73, 77, 82, 87, 88].

Pro výstup je rovněž dostupných několik tříd. Obecný výstupní proud (např. do souboru, do socketu, na obrazovku) představuje třída `java.io.OutputStream`. Pohodlnější způsob zapisování představuje např. jeden z jejích (nepřímých) potomků, třída `java.io.PrintStream`, která je typem statických atributů `out` a `err` (výchozí objekty představují standardní a chybový výstup) knihovní třídy `java.lang.System`. Její metody umožňují kromě zápisu řetězců i zápis primitivních datových typů nebo jiných objektů (voláním jejich metody `toString`), obojí s možností zakončení znakem nového řádku. Poskytuje i metodu pro výpis řetězce naformátovaného z předaného formátovacího řetězce a objektů představujících jeho argumenty [85, 86, 94].

2.7.2 C

Vstupní a výstupní operace jsou prováděny pomocí funkcí definovaných v hlavičkovém souboru `stdio.h`. Ke čtení ze standardního vstupu, jakož i k zápisu na standardní výstup existuje několik funkcí. Funkce `gets` a `getchar` umožňují jednoduché čtení po řádcích, resp. po jednotlivých znacích. Funkce `puts` a `putchar` umožňují jednoduchý zápis po řádcích, resp. po jednotlivých znacích [9].

Pro formátovaný vstup je k dispozici funkce `scanf`, která ze standardního vstupu načítá hodnoty podle formátovacího řetězce předaného jako první parametr funkce a ukládá je do proměnných, jejichž adresy přebírá jako další parametry funkce. Pro formátovaný výstup je k dispozici funkce `printf`, která analogickým způsobem vypisuje formátovaný řetězec, kdy prvním parametrem je řetězec ke zformátování a dalšími parametry jsou proměnné, jejichž hodnoty jsou do řetězce dosazeny [17, 19].

2.7.3 C++

Standardní vstup a výstup představuje objekt `std::cin`, resp. `std::cout`. Oba jsou definované v souboru `iostream`. První jmenovaný je instancí třídy `istream`, která používá operátor >> pro načtení znaků jako formátovaných dat do požadované proměnné. Jako alternativa existuje metoda `read`. Druhý jmenovaný objekt je instancí třídy `ostream`, ta používá operátor << pro zápis znaků jako formátovaných dat. Alternativou je metoda `write` [12, 22, 23].

2.7.4 C#

Standardní vstupní, výstupní a chybový výstupní proud představuje knihovní třída `System.Console` [38].

Její statické metody pro čtení umožňují načítání uživatelského vstupu po řádcích i po jednotlivých znacích. Takto načtený řetězec nebo znak lze převést na jiný základní datový typ (nebo obráceně) pomocí metod knihovní třídy `System.Convert`. Řetězec lze také převést na hodnotu jiného datového typu pomocí metody `Parse` nebo `TryParse` příslušného datového typu [33, 35, 39, 41, 42, 45, 49, 53, 54].

Statické metody třídy `System.Console` pro zápis umožňují zapisování jak řetězců, tak jiných datových typů (prostřednictvím voláním metody `ToString`), buď za sebou, nebo po řádcích. Umožňují i formátovaný zápis v duchu odpovídající metody třídy `PrintStream` v jazyku Java [38].

2.7.5 Pascal

Modul `System` definuje základní funkce pro vstupní a výstupní operace. Jsou jimi funkce `Read`, `ReadLn`, `Write` a `WriteLn`. Funkce `Read` slouží k načítání hodnot do proměnných předaných jako argumenty. Funkce `Write` slouží k zápisu hodnot proměnných předaných jako argumenty. Varianty s názvem končícím na řetězec `-Ln` slouží ke čtení po řádcích, resp. zápisu s ukončením řádky [107, 108, 114, 126, 127].

Funkce mohou mít jeden nebo dva parametry. V prvním případě parametr představuje proměnnou, do které má být vstup načten či která má být vypsána na výstup. V druhém případě je první parametr odkaz na soubor, se kterým se má pracovat místo standardního vstupu nebo výstupu [107, 108, 126, 127].

3 Vytvoření knihoven

Předmětem této kapitoly bude vytvoření knihoven obalujících funkce a metody uvedené v předchozí kapitole. Tyto knihovny budou poskytovat rozhraní ve stylu jazyka Java.

3.1 Obalové třídy základních datových typů

Budou vytvořeny knihovny reprezentující obalové třídy všech 8 primitivních datových typů jazyka Java: `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `Boolean` a `Character`. Implementovány budou následující části:

- metody `parse-` pro převod řetězcové reprezentace příslušného datového typu
- metody `toString` provádějící opačný proces
- konstanty definující hraniční hodnoty (jako `MIN_VALUE` a `MAX_VALUE`)

Podle potřeby budou implementovány další pomocné metody.

3.2 Třída String

Bude vytvořena knihovna představující třídu `String` s implementovanými metodami (příp. funkcemi): `equals`, `substring`, `indexOf` (přinejmenším pro hledání jednotlivých znaků), `length`, `trim`, `toLowerCase`, `toUpperCase`, `charAt`, `replace` (bez podpory regulárních výrazů), `startsWith`, `endsWith` a `compareTo`.

Knihovna bude umožňovat vytvoření objektu (příp. struktury) `String` pomocí standardního řetězce daného jazyka nebo pomocí pole jednobajtových znaků.

3.3 Třída StringTokenizer

Knihovna umožňující rozdělit řetězec na postupně načítané tokeny ve stylu třídy `StringTokenizer` bude implementovat dostupné metody: `countTokens`, `hasMoreTokens` a `nextToken` (alespoň variantu bez změny oddělovače).

3.4 Generická třída ArrayList<E>

Vytvořené knihovny budou zahrnovat implementaci dynamicky alokovaného pole v duchu třídy `ArrayList`. Minimální množina implementovaných metod bude zahrnovat: `add` (včetně vložení doprostřed seznamu), `get`, `size`, `isEmpty` a `toString`.

3.5 Generická třída LinkedList<E>

Vedle dynamicky alokovaného pole bude implementována rovněž knihovna představující obousměrný spojový seznam v duchu třídy `LinkedList`. Množina implementovaných metod se bude shodovat s tou, která bude dostupná pro třídu `ArrayList`.

3.6 Knihovní třída Arrays

Metody implementované v rámci knihovny `Arrays` pro manipulaci s poli budou následující: `equals`, `fill`, `copy`, `copyOfRange`, `sort`, `binarySearch` a `toString`.

3.7 Knihovní třída Collections

Množina metod, které budou implementovány v rámci knihovny `Collections` pro pokročilejší manipulaci se seznamy, bude pokrývat množinu stejnojmenných metod třídy `Arrays` s výjimkou metody `toString`, neboť obdobnou úlohu této metody pro seznamy plní jejich instanční metody `toString`.

3.8 Knihovní třída Math

V matematické knihovně `Math` bude dostupná implementace přinejmenším následujících metod: `abs`, `min`, `max`, `round`, `sqrt`, `pow`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `log` a `log10`. Případně budou definovány základní matematické konstanty π a e.

3.9 Knihovní třída System a třída PrintStream

Ze třídy `System` bude dostupná pouze reprezentace atributu `out` (objektu třídy `PrintStream`) a případně atributu `in` (pro účely napodobení volání kon-

struktoru třídy `Scanner` s předáním standardního vstupu představovaného objektem třídy `InputStream`). Implementované metody třídy `PrintStream` budou minimálně `print` a `println`, v případě jednoduché implementace i `format`. Řešen bude pouze zápis na standardní výstup.

3.10 Třída Scanner

Pro napodobení třídy `Scanner` bude vytvořena knihovna s implementací minimálně následujících metod pro načítání: `next`, `nextBoolean`, `nextByte`, `nextDouble`, `nextFloat`, `nextInt`, `nextLong`, `nextShort` a `nextLine`. Pro metody načítající čísla bude k dispozici přinejmenším implementace pro čtení číselných řetězců zapsaných v desítkové soustavě. Řešeno bude pouze čtení ze standardního vstupu.

4 Závěr

V rámci předmětu Projekt 5 byla dokončena analytická část bakalářské práce, představující první ze čtyř hlavních bodů zadání. Analýza spočívala v podrobnějším průzkumu základních tříd a metod jazyka Java a zjištění dostupnosti jejich potenciálních protějšků v jiných jazycích.

Ostatní body zadání, které představují realizační část bakalářské práce, byly rozpracovány. Podařilo se vytvořit podstatné části většiny knihoven s rozhraním ve stylu základních prozkoumaných tříd jazyka Java. Částečně nebo úplně vytvořenými knihovnami ve všech jazycích jsou v době psaní tohoto dokumentu knihovny `ArrayList`, `Arrays`, `Character`, `Collections`, `Integer`, `Math`, `PrintStream`, `Scanner`, `String` a `StringTokenizer`. Z největší části jsou hotové knihovny psané v jazyce C# díky velké podobnosti s jazykem Java, z nejmenší části prozatím v jazyce Pascal. Nicméně rozdíly v rozsahu dokončené práce pro jednotlivé jazyky nejsou příliš výrazné.

Dále se podařilo implementovat část aplikace pro správu zdrojových kódů vytvořených knihoven a rozpracovat jednotkové testy pro ověření správné funkčnosti těchto knihoven. Z aplikace je hotové grafické uživatelské rozhraní, byla navržena struktura XML dokumentů pro uchovávání zdrojových kódů a na základní úrovni implementováno několik operací s těmito kódy, zejména jejich zobrazení a vyhledávání ve vypsaném seznamu knihoven. Jednotkové testy byly prozatím rozpracovány v jazycích C# a Pascal, vytváření testů pro knihovny v jazycích C a C++ bude brzy následovat.

Splnění zbývajících bodů zadání vyžaduje dokončení knihoven, implementaci chybějící funkcionality aplikace pro správu a důkladné otestování aplikace i knihoven. Popis výše uvedeného spolu s výsledky již provedené analýzy bude obsahem závěrečného dokumentu k bakalářské práci.

Literatura

[1] *The GNU C Reference Manual* [online]. [cit. 1.12.2015]. Dostupné z: `http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html`.

[2] *abs - C++ Reference* [online]. [cit. 10.12.2015]. Dostupné z: `http://www.cplusplus.com/reference/cmath/abs/`.

[3] *Arrays - C++ Tutorials* [online]. [cit. 5.12.2015]. Dostupné z: `http://www.cplusplus.com/doc/tutorial/arrays/`.

[4] *bsearch - C++ Reference* [online]. [cit. 8.12.2015]. Dostupné z: `http://www.cplusplus.com/reference/cstdlib/bsearch/`.

[5] *calloc - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: `http://www.cplusplus.com/reference/cstdlib/calloc/`.

[6] *Character sequences - C++ Tutorials* [online]. [cit. 7.12.2015]. Dostupné z: `http://www.cplusplus.com/doc/tutorial/ntcs/`.

[7] *Classes (I) - C++ Tutorials* [online]. [cit. 1.12.2015]. Dostupné z: `http://www.cplusplus.com/doc/tutorial/classes/`.

[8] *<cmath> (math.h) - C++ Reference* [online]. [cit. 10.12.2015]. Dostupné z: `http://www.cplusplus.com/reference/cmath/`.

[9] *<cstdio> (stdio.h) - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: `http://www.cplusplus.com/reference/cstdio/`.

[10] *<cstdlib> (stdlib.h) - C++ Reference* [online]. [cit. 7.12.2015]. Dostupné z: `http://www.cplusplus.com/reference/cstdlib/`.

[11] *<cstring> (string.h) - C++ Reference* [online]. [cit. 7.12.2015]. Dostupné z: `http://www.cplusplus.com/reference/cstring/`.

[12] *<iostream> - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: http://www.cplusplus.com/reference/iostream/.

[13] *list - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/reference/list/list/.

[14] *malloc - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdlib/malloc/.

[15] *map - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/reference/map/map/.

[16] *Pointers - C++ Tutorials* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/doc/tutorial/pointers/.

[17] *printf - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdio/printf/.

[18] *qsort - C++ Reference* [online]. [cit. 8.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdlib/qsort/.

[19] *scanf - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: http://www.cplusplus.com/reference/cstdio/scanf/.

[20] *set - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/reference/set/set/.

[21] *binary_search - C++ Reference* [online]. [cit. 9.12.2015]. Dostupné z: http://www.cplusplus.com/reference/algorithm/-binary_search/.

[22] *cin - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: http://www.cplusplus.com/reference/iostream/cin/.

[23] *cout - C++ Reference* [online]. [cit. 11.12.2015]. Dostupné z: http://www.cplusplus.com/reference/iostream/cout/.

[24] *sort - C++ Reference* [online]. [cit. 9.12.2015]. Dostupné z: http://www.cplusplus.com/reference/algorithm/sort/.

[25] *stable_sort - C++ Reference* [online]. [cit. 9.12.2015]. Dostupné z: http://www.cplusplus.com/reference/algorithm/stable_sort/.

[26] *string - C++ Reference* [online]. [cit. 7.12.2015]. Dostupné z: http://www.cplusplus.com/reference/string/-string/?kw=string.

[27] *<string> - C++ Reference* [online]. [cit. 7.12.2015]. Dostupné z: http://www.cplusplus.com/reference/string/.

[28] *Variables and types - C++ Tutorials* [online]. [cit. 3.12.2015]. Dostupné z: http://www.cplusplus.com/doc/tutorial/variables/.

[29] *vector - C++ Reference* [online]. [cit. 5.12.2015]. Dostupné z: http://www.cplusplus.com/reference/vector/vector/.

[30] *Array – třída (System)* [online]. [cit. 6.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.array(v=vs.110).aspx.

[31] *Arrays Tutorial (C#)* [online]. [cit. 6.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/aa288453(v=vs.71).aspx.

[32] *Visual C# Language (C#)* [online]. [cit. 2.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/aa287558(v=vs.71).aspx.

[33] *bool (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/c8f5xwh7.aspx.

[34] *byte (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/5bdb6693.aspx.

[35] *char (C# Reference)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/x9h8tsay.aspx.

[36] *Common Language Runtime (CLR)* [online]. [cit. 2.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/8bs2ecf4(v=vs.110).aspx.

[37] *Comparison(T) – delegát (System)* [online]. [cit. 9.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/tfakywbh(v=vs.110).aspx.

[38] *Console – třída (System)* [online]. [cit. 12.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.console(v=vs.110).aspx.

[39] *Convert – třída (System)* [online]. [cit. 12.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.convert(v=vs.110).aspx.

[40] *decimal (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/364x0z75.aspx.

[41] *double (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/678hzkk9.aspx.

[42] *float (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/b1e65aza.aspx.

[43] *IComparable(T) – rozhraní (System)* [online]. [cit. 9.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/4d7sx9hd(v=vs.110).aspx.

[44] *IComparer(T) – rozhraní (System.Collections.Generic)* [online]. [cit. 9.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/8ehhxeaf(v=vs.110).aspx.

[45] *int (C# Reference)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/5kzh1b5w.aspx.

[46] *Compiling MSIL to Native Code* [online]. [cit. 2.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/ht8ecch6(v=vs.100).aspx.

[47] *LinkedList(T) – třída (System.Collections.Generic)* [online]. [cit. 6.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/he2s3bh7(v=vs.110).aspx.

[48] *List(T) – třída (System.Collections.Generic)* [online]. [cit. 6.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/6sh2ey19(v=vs.110).aspx.

[49] *long (C# Reference)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/ctetwysk.aspx.

[50] *Math – třída (System)* [online]. [cit. 10.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.math(v=vs.110).aspx.

[51] *Compiling to MSIL* [online]. [cit. 2.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/c5tkafs1(v=vs.100).aspx.

[52] *object (C# Reference)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/9kkx3h3c.aspx.

[53] *sbyte (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/d86he86x.aspx.

[54] *short (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/ybs77ex4.aspx.

[55] *string (C# Reference)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/362314fe.aspx.

[56] *String Class (System)* [online]. [cit. 7.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.string(v=vs.110).aspx.

[57] *String.Split Method (String[], StringSplitOptions) (System)* [online]. [cit. 7.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/tabh47cf(v=vs.110).aspx.

[58] *Built-in Data Types* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/cs7y5x0x(v=vs.90).aspx.

[59] *uint (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/x0sksh43.aspx.

[60] *ulong (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/t98873t4.aspx.

[61] *ushort (Referenční dokumentace jazyka C#)* [online]. [cit. 3.12.2015]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/cbf1574z.aspx.

[62] *Arrays (The Java^{TM} Tutorials > Learning the Java Language > Language Basics)* [online]. [cit. 5.12.2015]. Dostupné z: `https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html`.

[63] *ArrayList (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: `https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html`.

[64] *Arrays (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html`.

[65] *Boolean (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: `https://docs.oracle.com/javase/7/docs/api/java/lang/Boolean.html`.

[66] *Byte (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: `https://docs.oracle.com/javase/7/docs/api/java/lang/Byte.html`.

[67] *Character (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: `https://docs.oracle.com/javase/7/docs/api/java/lang/Character.html`.

[68] *Collection (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: `https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html`.

[69] *Collections (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html`.

[70] *Comparable (Java Platform SE 7)* [online]. [cit. 8.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html`.

[71] *Comparator (Java Platform SE 7)* [online]. [cit. 8.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html`.

[72] *Double (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: `https://docs.oracle.com/javase/7/docs/api/java/lang/Double.html`.

[73] *Float (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/-java/lang/Float.html.

[74] *HashMap (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/-java/util/HashMap.html.

[75] *HashSet (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/-java/util/HashSet.html.

[76] *InputStream (Java Platform SE 7)* [online]. [cit. 11.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/-java/io/InputStream.html.

[77] *Integer (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/-java/lang/Integer.html.

[78] *java* [online]. [cit. 1.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/-technotes/tools/windows/java.html.

[79] *Java SE 7 Java Compiler (javac)-related APIs and Developer Guides* [online]. [cit. 1.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/-technotes/guides/javac/index.html.

[80] *LinkedList (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/-java/util/LinkedList.html.

[81] *List (Java Platform SE 7)* [online]. [cit. 5.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/-java/util/List.html.

[82] *Long (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/-java/lang/Long.html.

[83] *Math (Java Platform SE 7)* [online]. [cit. 10.12.2015]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/-java/lang/Math.html.

[84] *Object (Java Platform SE 7)* [online]. [cit. 1.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/-java/lang/Object.html`.

[85] *OutputStream (Java Platform SE 7)* [online]. [cit. 11.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/-java/io/OutputStream.html`.

[86] *PrintStream (Java Platform SE 7)* [online]. [cit. 11.12.2015]. Dostupné z: `https://docs.oracle.com/javase/7/docs/api/-java/io/PrintStream.html`.

[87] *Scanner (Java Platform SE 7)* [online]. [cit. 11.12.2015]. Dostupné z: `https://docs.oracle.com/javase/7/docs/api/-java/util/Scanner.html`.

[88] *Short (Java Platform SE 7)* [online]. [cit. 3.12.2015]. Dostupné z: `https://docs.oracle.com/javase/7/docs/api/-java/lang/Short.html`.

[89] *String (Java Platform SE 7)* [online]. [cit. 7.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/-java/lang/String.html`.

[90] *StringBuffer (Java Platform SE 7)* [online]. [cit. 7.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/-java/lang/StringBuffer.html`.

[91] *StringBuilder (Java Platform SE 7)* [online]. [cit. 7.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/-java/lang/StringBuilder.html`.

[92] *Strings (The Java™ Tutorials > Learning the Java Language > Numbers and Strings)* [online]. [cit. 6.12.2015]. Dostupné z: `https://docs.oracle.com/javase/tutorial/-java/data/strings.html`.

[93] *StringTokenizer (Java Platform SE 7)* [online]. [cit. 7.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/-java/util/StringTokenizer.html`.

[94] *System (Java Platform SE 7)* [online]. [cit. 11.12.2015]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/-java/lang/System.html`.

[95] *Primitive Data Types (The Java™ Tutorials > Learning the Java Language > Language Basics)* [online]. [cit. 3.12.2015]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html.

[96] *Java SE 7 Virtual Machine (VM)-related APIs & Developer Guides* [online]. [cit. 1.12.2015]. Dostupné z: http://docs.oracle.com/javase/7/docs/technotes/guides/vm/index.html.

[97] *About the Pascal language* [online]. [cit. 2.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refli6.html#x76000.

[98] *Char or AnsiChar* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu7.html#x30330003.2.1.

[99] *Single-byte String types* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu10.html#x33380003.2.4.

[100] *Base types* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refse12.html#x26250003.1.

[101] *Ordinal types* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu5.html#x27280003.1.1.

[102] *Character types* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refse13.html#x29320003.2.

[103] *Arrays* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu15.html#x39520003.3.1.

[104] *Ordinal types* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu5.html#x27260003.1.1.

[105] *Reference for unit 'math': Procedures and functions* [online]. [cit. 10.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/math/index-5.html.

[106] *Pointers* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refse15.html#x43-610003.4.

[107] *Read* [online]. [cit. 12.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/system/read.html.

[108] *ReadLn* [online]. [cit. 12.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/system/readln.html.

[109] *Real types* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu6.html#x28-310003.1.2.

[110] *Single-byte String types* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu10.html#x33-370003.2.4.

[111] *Arrays* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu15.html#x39-510003.3.1.

[112] *Single-byte String types* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu10.html#x33-360003.2.4.

[113] *Structured Types* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refse14.html#x38-480003.3.

[114] *Reference for unit 'System': Procedures and functions* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/system/index-5.html.

[115] *TFPGList* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/fgl/tfpglist.html.

[116] *TFPGObjectList* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/fgl/tfpgobjectlist.html.

[117] *TFPList* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/classes/tfplist.html.

[118] *TFPList.Sort* [online]. [cit. 9.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/classes/tfplist.sort.html.

[119] *TList* [online]. [cit. 6.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/classes/tlist.html.

[120] *TList.Sort* [online]. [cit. 9.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/classes/tlist.sort.html.

[121] *TListSortCompare* [online]. [cit. 9.12.2015]. Dostupné z: http://www.freepascal.org/docs-html-3.0.0/rtl/classes/tlistsortcompare.html.

[122] *TStringList* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/classes/tstringlist.html.

[123] *TStrings* [online]. [cit. 7.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/classes/tstrings.html.

[124] *3 Types* [online]. [cit. 3.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refch3.html#x25-240003.

[125] *WideChar* [online]. [cit. 4.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/ref/refsu8.html#x31-340003.2.2.

[126] *Write* [online]. [cit. 12.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/system/write.html.

[127] *WriteLn* [online]. [cit. 12.12.2015]. Dostupné z: http://www.freepascal.org/docs-html/rtl/system/writeln.html.

[128] HEROUT, P. *Učebnice jazyka C*. KOPP, 1994. ISBN 80-85828-21-9.

[129] PRATA, S. *Mistrovsví v C++*. Computer Press, 2001. ISBN 80-7226–339-0.

[130] SATRAPA, P. *Pascal pro zelenáče*. Neocortex, 2000. ISBN 80-86330-03-6.

[131] SHARON ZAKHOUR, J. R. I. R. T. R. M. H. S. H. *Java 6 Výukový kurz*. Computer Press, 2007. ISBN 978-80-251-1575-6.

[132] SHARP, J. *Microsoft Visual C# 2010 Krok za krokem*. Computer Press, 2010. ISBN 978-80-251-3147-3.