



**FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI**

Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Semestrální práce z programování v jazyce C

Generátor dokumentace zdrojového kódu

Štěpán Lukeš
A21B0625P
slukes@students.zcu.cz

7. 1. 2024

Obsah

1	Zadání	2
2	Analýza úlohy	5
2.1	Rozbor problémů	5
2.2	Možné řešení	5
2.2.1	Lex	5
2.2.2	Yacc	6
2.2.3	Spojový seznam	6
2.2.4	Zásobník	6
2.2.5	Metoda přímého zpracování a analýzy	7
3	Implementace	8
3.1	Popis implementace	8
3.2	Moduly	8
3.3	Struktury	9
3.3.1	Struktura DocComment	10
3.3.2	Struktura FunctionDoc	11
3.4	Funkce	11
4	Uživatelská příručka	15
4.1	Návod k použití	15
4.2	Průběh programu	15
4.3	Ukázky výstupu programu	18
5	Závěr	21
6	Zdroje	22

1 Zadání

Naprogramujte v ANSI C přenositelnou **konzolovou aplikaci**, která prohledá vstupní soubor se zdrojovým kódem v programovacím jazyce ANSI C a na základě tam nalezených speciálních značek vygeneruje výstupní soubor s dokumentací vstupního zdrojového kódu ve formátu \LaTeX . Vstupem aplikace bude tedy textový soubor se zdrojovým kódem v programovacím jazyce ANSI C opatřeným tzv. *dokumentačními komentáři* uvedenými speciálními značkami podle níže uvedené specifikace. Aplikace tyto značky vyhledá a zpracuje a následně z informací v nich obsažených vygeneruje dokumentaci předaného vstupního zdrojového kódu. Výstupem aplikace bude pak zdrojový soubor ve značkovacím jazyce \TeX , který půjde přeložit typografickým systémem \LaTeX . Úloha je velmi podobná (ačkoli výrazně zjednodušená) existujícím projektům JavaDoc, **Doxygen**, **PasDoc**, apod. Tyto projekty můžete v rámci přípravy k řešení semestrální práce prozkoumat a volně se jimi inspirovat.

Program se bude spouštět příkazem

```
ccdoc.exe <vstupní soubor[.c|.h]> [<výstupní soubor[.tex]>]
```

Symbol $\langle \text{vstupní soubor} \rangle$ zastupuje povinný parametr – název vstupního souboru se zdrojovým kódem v jazyce ANSI C (může to být jak `.c`, tak hlavičkový soubor `.h`). Symbol $\langle \text{výstupní-soubor} \rangle$ pak představuje nepovinný parametr, kterým je jméno souboru s výstupem programu. Není-li druhý, nepovinný, parametr uveden, směřujte výstup programu do souboru pojmenovaného stejným jménem, jaký má vstupní soubor, ke kterému je připojen suffix `-doc` a přípona `.tex` a tento soubor bude umístěný ve stejné složce, kde se nachází vstupní soubor.

Úkolem Vámi vyvinutého programu tedy bude:

1. Načíst vstupní soubor se zdrojovým kódem v jazyce ANSI C.
2. Projít tento vstupní soubor, nalézt v něm připojení všech lokálních hlavičkových souborů (příkazy preprocesoru `#include "..."`) a projít i všechny takto připojené lokální hlavičkové soubory a jim odpovídající soubory zdrojové (tj. je-li připojen hlavičkový soubor `mylib.h`, je třeba hledat také odpovídající zdrojový soubor `mylib.c`; uvědomte si, že i tyto soubory mohou obsahovat příkazy preprocesoru `#include "..."`).

3. Ve všech vstupních souborech (i těch, které nejsou předané přímo na příkazové řádce, ale jsou importovány prostřednictvím příkazu preprocesoru `#include "..."`) nalézt dokumentační komentáře (specifikované níže).
4. Všechny nalezené komentáře zpracovat – zajistit jejich případné spojení (protože funkce může být okomentována dokumentačním komentářem jak v hlavičkovém souboru, tak také v souboru `.c`) – a vypsát jim odpovídající výstup do souboru ve formátu `LaTeX` podle pokynů uvedených dále v textu tohoto zadání.

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

```
ccdoc.exe "E:\My Work\Test\test 01.c"test01-doc.tex
```

Nebo takto:

```
ccdoc.exe "E:\My Work\Test\test01.c"doc\test01-doc.tex
```

Nebo pouze takto:

```
ccdoc.exe work_md1.c
```

Druhý z uvedených příkladů spuštění by měl vést k vytvoření výstupního souboru `work_md1-doc.tex` v tomtéž adresáři, kde se nachází zdrojový soubor `work_md1.c`

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

```
ccdoc.exe /home/user/work/test01.c test01-doc.tex
```

Uvažujte všechny možné specifikace jak vstupního, tak výstupního souboru, které jsou přípustné v daném operačním systému. Pokud nebude na příkazové řádce uveden alespoň jeden argument (tj. jméno souboru se zdrojovým kódem), vypíše chybové hlášení a stručný návod k použití programu (v angličtině).

Hotovou práci odevzdejte v jediném archivu typu ZIP prostřednictvím automatického odevzdávacího a validačního systému. Postupujte podle instrukcí uvedených na webu předmětu. Archiv nechtě obsahuje všechny zdrojové soubory potřebné k přeložení programu, **makefile** pro Windows i Linux (pro překlad v Linuxu připravte soubor pojmenovaný **makefile** a pro Windows

`makefile.win`) a dokumentaci ve formátu PDF vytvořenou v typografickém systému \TeX , resp. \LaTeX . Bude-li některá z částí chybět, kontrolní skript Vaši práci odmítne.

Celé zadání semestrální práce je k dispozici zde: <https://www.kiv.zcu.cz/studies/predmety/pc/data/works/sw2023-01.pdf>

2 Analýza úlohy

2.1 Rozbor problémů

V semestrálním projektu pro generování dokumentace zdrojového kódu v jazyce ANSI C ve formátu \LaTeX byly identifikovány následující hlavní problémy:

1. **Lexikální Analýza a Zpracování Textu:** Základním problémem je efektivní zpracování a analýza zdrojového kódu. Toto zahrnuje rozpoznání speciálních dokumentačních komentářů, což vyžaduje spolehlivou lexikální analýzu a schopnost zpracovávat texty různých formátů.
2. **Formátování a Konverze do \LaTeX formátu:** Další výzvou je převod textových komentářů na strukturovaný \LaTeX formát. Toto vyžaduje detailní znalosti \LaTeX syntaxe a efektivní metody pro převod běžného textu na formátovaný dokument.
3. **Přenositelnost a Kompatibilita:** Zajištění, že aplikace bude funkční a konzistentní napříč různými operačními systémy, je klíčové pro širokou použitelnost nástroje. To vyžaduje pečlivé testování a optimalizaci pro různé platformy.

2.2 Možné řešení

2.2.1 Lex

Jedním z možných řešení by mohl být nástroj **Lex**. **Lex** by se v tomto projektu využil k identifikaci a extrakci dokumentačních komentářů. Specifikace v **Lex** by definovala pravidla pro rozpoznávání komentářů, které začínají `/**` nebo `/*!`, a následně by extrahovala relevantní text z těchto komentářů. Tento proces by zahrnoval lexikální analýzu zdrojového kódu a převod nalezených komentářů na řadu tokenů, které by pak byly dále zpracovány pro převod do formátu \LaTeX . Takový přístup by umožnil efektivní zpracování a strukturování dokumentace a zároveň by zajistil přesnost při identifikaci relevantních částí kódu pro dokumentaci. Nicméně, zvládnutelnost tohoto přístupu vyžaduje určité znalosti a zkušenosti s lexikálními analyzátory a regulárními výrazy.

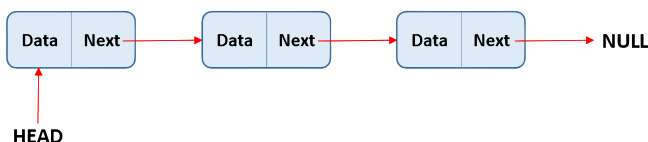
2.2.2 Yacc

Další možné řešení pro tento projekt se nabízí nástroj **Yacc** (Yet Another Compiler Compiler). **Yacc** by byl použit k syntaktické analýze extrahovaných dokumentačních komentářů a jejich převodu na strukturovaný \LaTeX formát. Tento přístup by umožnil efektivní zpracování komplexních gramatických struktur v dokumentaci a jejich transformaci do přesně formátovaného dokumentu. Nicméně, využití **Yacc** vyžaduje znalosti tvorby gramatik a syntaktických analyzátorů. Integrace **Yacc** s lexikálním analyzátozem by mohla být technicky velmi náročná.

2.2.3 Spojový seznam

Spojový seznam by mohl být využit jako efektivní struktura pro organizaci a uchování extrahovaných dokumentačních komentářů. Jeho flexibilita v přidávání a odebírání prvků by umožnila dynamické spravování obsahu dokumentace, což je zvláště užitečné při zpracování velkých datových sad. Využití spojového seznamu však vyžaduje pečlivé plánování struktury dat a může přinést výzvy v přístupu k jednotlivým prvkům.

Jak spojový seznam funguje znázorňuje následující obrázek 2.1:

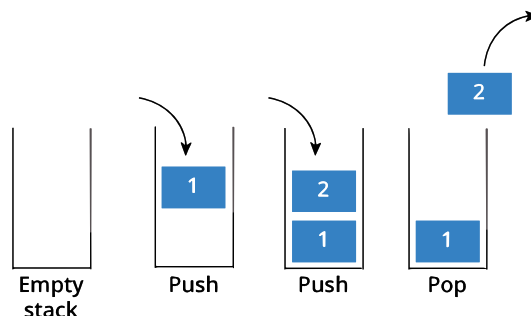


Obrázek 2.1: Struktura spojového seznamu

2.2.4 Zásobník

Zásobník by se hodil pro dočasné uchovávání a následné zpracování extrahovaných dat v procesu generování dokumentace. Jeho LIFO (Last In, First Out) vlastnost by mohla být efektivní pro určité aspekty zpracování, jako je například sledování a zpracování vnořených struktur v dokumentaci. Nicméně, správa zásobníku může být složitější v kontextu vícevrstevných nebo složitě strukturovaných dat.

Jak zásobník funguje znázorňuje následující obrázek 2.2:



Obrázek 2.2: Struktura zásobníku

2.2.5 Metoda přímého zpracování a analýzy

Další metodou je přímé zpracování a analýza zdrojového kódu. Tato metoda spočívá v manuální implementaci analýzy zdrojového kódu, bez použití nástrojů jako **Lex** nebo **Yacc**. Přímé zpracování zahrnuje čtení a interpretaci kódu řádek po řádku, identifikaci komentářů a extrakci relevantních informací. Tento přístup byl zvolen kvůli jeho přizpůsobivosti a kontrole nad procesem extrakce a formátování dokumentace. Vhodná je zejména pro projekty, kde je potřeba specifické zpracování nebo přizpůsobení výstupu dokumentace. Výběr této metody je také podložen snahou o jednoduchost a přímou kontrolu nad zpracováním bez potřeby zvládnutí složitějších nástrojů nebo jazyků.

3 Implementace

3.1 Popis implementace

V rámci implementace projektu pro generování dokumentace zdrojového kódu v ANSI C do formátu \LaTeX je zaměření na přímé zpracování a analýzu zdrojového kódu. Implementace zahrnuje funkce pro zpracování souborů, identifikaci dokumentačních komentářů a extrakci z nich relevantních informací, které jsou následně formátovány do souboru \LaTeX . Program využívá vlastní metody pro analýzu syntaxe a struktury kódu, včetně funkcí pro manipulaci s textem a datovými strukturami.

Modulární struktura programu s jasně definovanými funkcemi pro jednotlivé části procesu generování dokumentace zajišťuje přehlednost a efektivitu.

3.2 Moduly

Každý modul má specifickou funkci a společně přispívají k efektivní implementaci celého procesu generování dokumentace.

Modul `global.h`

Definuje globální proměnné a struktury použité v celém projektu. Tento modul zajišťuje centrální přístup k důležitým datovým strukturám a proměnným. Modul je zde závislý na modulech `constants.h` a `data_structures.h`

Modul `constants.h`

Obsahuje konstanty, které se využívají napříč celým projektem. Tento modul slouží k udržení konzistence hodnot používaných v různých částech kódu.

Modul `data_structures.h/c`

Obsahuje definice a implementace datových struktur pro správu informací o dokumentačních komentářích a funkcích v projektu. Tyto struktury jsou klíčové pro efektivní manipulaci s daty během zpracování a generování dokumentace. `data_structures.c` je zde závislý na `data_structures.h`. Zároveň `data_structures.h` je závislý na modulu `constants.h`.

Modul **utility.c/h**

Poskytuje pomocné funkce pro manipulaci s textem, například pro odstranění bílých znaků nebo pro escapování speciálních znaků pro \LaTeX . Zde modul **utility.c** je závislý na stejnojmenném modulu **utility.h**.

Modul **file__processing.c/h**

Zabývá se zpracováním vstupních souborů, identifikací dokumentačních komentářů a extrakcí relevantních informací z těchto komentářů. Modul **file__processing.c** spoléhá na moduly **file__processing.h**, **global.h**, **constants.h** a **documentation__processing.h**. Následně modul **file__processing.h** je závislý na modulu **data__structures.h**.

Modul **documentation__processing.c/h**

Zaměřuje se na zpracování extrahovaných dokumentačních komentářů a jejich přípravu pro formátování v \LaTeX u. Modul **documentation__processing.c** je závislý na modulu **documentation__processing.h**. A modul **documentation__processing.h** je závislý na modulu **data__structures.h**.

Modul **latex__formatting.c/h**

Stará se o finální formátování extrahovaných informací do formátu \LaTeX včetně správné syntaxe a strukturování dokumentu. Modul **latex__formatting.c** je závislý na modulech **latex__formatting.h** a **global.h**. V případě modulu **latex__formatting.h** je tato závislost podmíněná modulem **data__structures.h**.

Modul **main.c**

Hlavní spouštěcí bod programu, řídí celkový proces generování dokumentace, od načtení vstupních souborů až po výstup finálního \LaTeX dokumentu. Spoléhá na funkce a struktury definované v ostatních modulech – konkrétně v modulech **global.h**, **file__processing.h** a **latex__formatting.h**.

3.3 Struktury

Program obsahuje 2 struktury. Strukturu **DocComment** a strukturu **FunctionDoc**.

3.3.1 Struktura DocComment

Uchovává informace o jednotlivých dokumentačních komentářích v zdrojovém kódu, včetně stručných popisů, detailních informací a parametrů.

```
typedef struct {  
    char brief[1024];  
    char details[4096];  
    char freeText[MAX_TEXT_LENGTH];  
    char lastLineHadText;  
    char paramName[MAX_PARAMS][256];  
    char paramDesc[MAX_PARAMS][1024];  
    int paramCount;  
    char returnVal[256];  
    char author[256];  
    char version[256];  
} DocComment;
```

Struktura DocComment obsahuje následující atributy:

- **brief** – krátký popis
- **details** – detailní popis
- **freeText** – dodatečný text nebo poznámky
- **lastLineHadText** – indikátor, zda poslední řádek obsahoval text
- **paramName** – pole názvů parametrů
- **paramDesc** – pole popisů parametrů
- **paramCount** – počet parametrů
- **returnVal** – popis návratové hodnoty
- **author** – autor
- **version** – verze

3.3.2 Struktura FunctionDoc

Slouží k reprezentaci dokumentace konkrétní funkce, včetně jejího názvu, návratového typu, prototypu a souvisejících dokumentačních komentářů.

```
typedef struct {
    char returnType[256];
    char functionName[256];
    char moduleName[256];
    char prototype[1024];
    char fileTypes[2];
    DocComment comment;
} FunctionDoc;
```

Struktura FunctionDoc obsahuje následující atributy:

- **returnType** – návratový typ funkce
- **functionName** – jméno funkce
- **moduleName** – modul, ve kterém se funkce nachází
- **prototype** – prototyp funkce
- **fileTypes** – typy souborů (hlavičkový/zdrojový)
- **comment** – přidružený dokumentační komentář ('DocComment')

3.4 Funkce

Program celkem obsahuje 13 pomocných funkcí a následně hlavní funkci main. Všechny funkce jsou patřičně popsány a důležité funkce jsou popsány obsáhleji.

Funkce void initDocComment(DocComment *comment)

Nastavuje všechna pole v struktuře **DocComment** na výchozí hodnoty (prázdné řetězce nebo nuly), aby byla připravena pro nový dokumentační komentář.

Funkce bool fileAlreadyProcessed(const char *filename)

Kontroluje, zda se jméno souboru již nachází v poli zpracovaných souborů, což zabrání opakovanému zpracování tohoto souboru.

Funkce void addFileToProcessed(const char *filename)

Přidává jméno souboru do pole zpracovaných souborů, které se používá pro sledování již zpracovaných souborů.

Funkce void trimLine(const char *input, char *output)

Odstraňuje bílé znaky na začátku a konci zadaného vstupního řetězce a výsledný upravený řetězec ukládá do výstupního parametru.

Funkce void processComment(const char *line, DocComment *comment)

Funkce zpracovává řádek kódu a extrahuje z něj informace pro dokumentační komentář. Na začátku odstraňuje přebytečné bílé znaky a znaky '*', které jsou typické pro komentáře ve stylu C. Poté hledá speciální značky (např. '@brief', '@param', '@return'), které určují typ informace v komentáři. Každá značka spouští kód, který ukládá odpovídající informaci do struktury **DocComment**. Výsledkem jsou správně zpracované různé části komentářů, jako jsou stručné popisy, parametry a návratové hodnoty.

Funkce void processFile(const char *filename, FILE *outputFile)

Slouží k zpracování zdrojového souboru a generování dokumentace. Tato funkce nejprve ověří, zda již byl daný soubor zpracován, pokud ano, přeskočí jej. Pokud ne, otevře soubor pro čtení. Pokud se soubor nepodaří otevřít, funkce skončí s chybovým hlášením. Funkce poté čte soubor řádek po řádku a vyhledává dokumentační komentáře, začínající '/*' nebo '/*!'. Po nalezení komentáře, funkce inicializuje strukturu **DocComment** a vstupuje do režimu zpracování komentáře. Jakmile narazí na ukončující komentář '*/', připraví komentář pro další zpracování. Funkce také zpracovává deklarace funkcí, extrahuje z nich informace a ukládá je do struktury **FunctionDoc**. Pokud soubor obsahuje hlavičkový soubor, pokusí se najít a zpracovat odpovídající zdrojový soubor. Nakonec soubor zavře. Funkce také zpracovává vnořené **#include** direktivy pro zahrnutí dokumentace z dalších souborů.

Funkce bool addFunctionDoc(FunctionDoc *funcDoc)

slouží k přidání dokumentace k funkci do globálního pole dokumentovaných funkcí **functionDocs**. Nejprve provádí kontrolu, zda již existuje dokumentace pro funkci se stejným návratovým typem a jménem. Pokud ano, aktualizuje typy souborů a sloučí komentáře pomocí funkce **mergeDocComments**.

Pokud dokumentace pro danou funkci neexistuje, přidává novou dokumentaci do pole. Funkce vrátí **'true'** v případě úspěšného přidání nebo aktualizace dokumentace, jinak **'false'**. Tato funkce je klíčová pro správné shromažďování a organizaci dokumentace funkcí napříč různými soubory projektu.

Funkce void mergeDocComments(DocComment *dest, const DocComment *src)

slouží k sloučení dvou komentářů. Pokud má zdrojový komentář (**src**) nějaké neprázdné pole, které v cílovém komentáři (**dest**) není vyplněno, překopíruje se hodnota do cílového komentáře. Toto zahrnuje pole jako **brief**, **details**, **returnVal**, **author**, a **version**. Funkce také sloučí pole parametrů, pokud existují v zdrojovém komentáři a nejsou již zahrnuty v cílovém.

Funkce void formatModuleName(char *moduleName)

Upravuje název modulu pro jeho použití v souboru \LaTeX .

Funkce void escapeLaTeXChars(char *str)

Escapuje speciální znaky pro jejich správné zobrazení v souboru \LaTeX .

Funkce void startLaTeXDocument(FILE *outputFile)

Zahajuje dokument \LaTeX s počátečními definicemi.

Funkce void formatToLaTeX(const FunctionDoc *funcDoc, FILE *outputFile, bool *isNewModule, char *lastModuleName)

Tato funkce má klíčovou roli v převádění dokumentačních dat do formátu \LaTeX . Začíná kontrolou, zda je zpracováván nový modul, a podle toho vkládá sekci do \LaTeX dokumentu. Následně formátuje a zapíše do dokumentu informace jako název modulu, prototyp funkce, stručný popis, argumenty, návratovou hodnotu, podrobný popis, autora a verzi. Každý aspekt je pečlivě formátován s ohledem na syntaxi \LaTeX . Speciální znaky jsou upraveny pro správné zobrazení v \LaTeX . Tato funkce je nezbytná pro převod extrahovaných informací do strukturované a přesně formátované dokumentace.

Funkce void endLaTeXDocument(FILE *outputFile)

Ukončuje \LaTeX dokument.

Funkce `int main(int argc, char *argv[])`

Funkce **main** je vstupním bodem programu. Začíná kontrolou, zda byl poskytnut dostatečný počet argumentů (jména souborů). Pokud ne, vypíše chybovou zprávu se stručným návodem v angličtině. Poté se pokouší otevřít vstupní soubor, a pokud soubor neexistuje nebo se nepodaří jej otevřít, vypíše další chybovou zprávu. Následně se přistupuje k zpracování vstupního souboru prostřednictvím funkce **processFile()**. Na konci se generuje dokument ve formátu \LaTeX , voláním funkcí **startLaTeXDocument()**, **formatToLaTeX()** pro každou dokumentovanou funkci a **endLaTeXDocument()** pro dokončení dokumentace. Funkce vrací hodnotu 0, pokud proběhne vše bez chyby, nebo návratový kód chyby, pokud dojde k nějakému problému.

4 Uživatelská příručka

4.1 Návod k použití

Aplikace je konzolová a před spuštěním je nutné ji nejprve přeložit. O přeložení aplikace se stará soubor *makefile*, který se spustí příkazem *make* v konzoli. Pro použití *make* ve Windows je třeba nainstalovat nástroj, který tento příkaz podporuje, jako je například MinGW. Když je aplikace úspěšně přeložena, je možné ji spustit.

Pro operační systém Windows se aplikace spouští příkazem:

```
ccdoc.exe <vstupní soubor[.c|.h]> [<výstupní soubor[.tex]>]
```

Alternativně pro UNIXový operační systém se aplikace spouští příkazem:

```
./ccdoc.exe <vstupní soubor[.c|.h]> [<výstupní soubor[.tex]>]
```

Symbol *<vstupní soubor>* představuje první, povinný, parametr – název vstupního souboru se zdrojovým kódem v jazyce ANSI C (může to být jak zdrojový soubor – tedy s příponou *.c*, tak hlavičkový soubor – tedy s příponou *.h*). Symbol *<výstupní-soubor>* pak představuje druhý, nepovinný, parametr, kterým je jméno souboru s výstupem aplikace. Není-li druhý, nepovinný, parametr uveden, výstup aplikace bude vytvořen do souboru pojmenovaného stejným jménem, jaký má vstupní soubor, ke kterému je připojen suffix *-doc* a přípona *.tex* a tento soubor bude umístěn ve stejné složce, kde se nachází vstupní soubor.

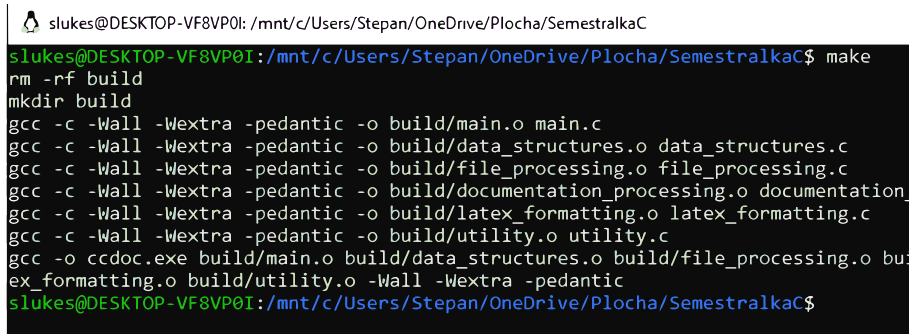
Pokud aplikace neobdrží požadovaný počet argumentů na příkazovém řádku či pokud argumenty jsou chybné (neexistující soubory, nepovolené znaky v jejich názvech, apod.), vypíše aplikace příslušné chybové hlášení v angličtině.

4.2 Průběh programu

Po zadání vstupních parametrů aplikace vytvoří zdrojový soubor programátorské dokumentace v makrojazyce *L^AT_EX*.

Následující obrázky ilustrují klíčové fáze průběhu aplikace. Inicializace začíná přeložením aplikace (viz obrázek 4.1), po kterém následuje podání vstupního souboru aplikaci jako prvního parametru (obrázek 4.2). Spuštění aplikace je

zobrazeno na obrázku 4.3. Dále je vygenerován výstupní soubor ve formátu L^AT_EX (obrázek 4.4). V neposlední řadě je zde náhled odpovídajícího souboru ve formátu PDF (obrázek 4.5).

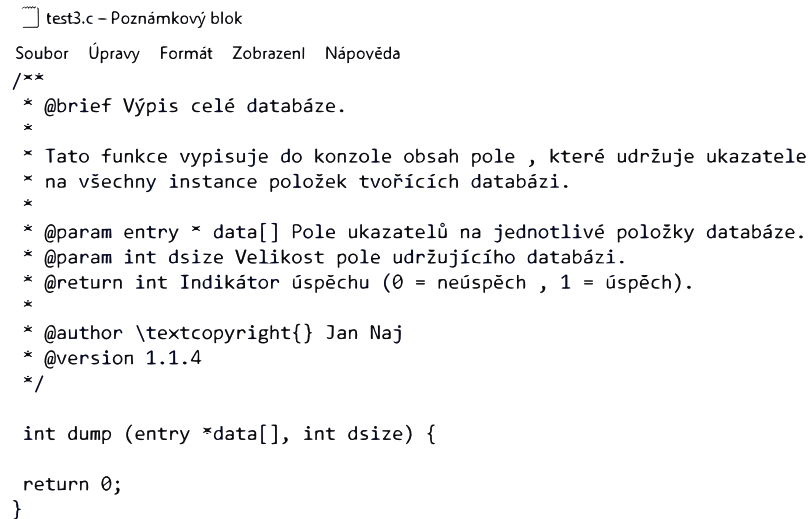


```

slukes@DESKTOP-VF8VP0I: /mnt/c/Users/Stepan/OneDrive/Plocha/SemestralkaC
slukes@DESKTOP-VF8VP0I:/mnt/c/Users/Stepan/OneDrive/Plocha/SemestralkaC$ make
rm -rf build
mkdir build
gcc -c -Wall -Wextra -pedantic -o build/main.o main.c
gcc -c -Wall -Wextra -pedantic -o build/data_structures.o data_structures.c
gcc -c -Wall -Wextra -pedantic -o build/file_processing.o file_processing.c
gcc -c -Wall -Wextra -pedantic -o build/documentation_processing.o documentation_
gcc -c -Wall -Wextra -pedantic -o build/latex_formatting.o latex_formatting.c
gcc -c -Wall -Wextra -pedantic -o build/utility.o utility.c
gcc -o ccdoc.exe build/main.o build/data_structures.o build/file_processing.o bui
ex_formatting.o build/utility.o -Wall -Wextra -pedantic
slukes@DESKTOP-VF8VP0I:/mnt/c/Users/Stepan/OneDrive/Plocha/SemestralkaC$

```

Obrázek 4.1: Přeložení aplikace příkazem *make*



```

test3.c - Poznámkový blok
Soubor Úpravy Formát Zobrazení nápověda
/**
 * @brief Vypis celé databáze.
 *
 * Tato funkce vypisuje do konzole obsah pole , které udržuje ukazatele
 * na všechny instance položek tvořících databázi.
 *
 * @param entry * data[] Pole ukazatelů na jednotlivé položky databáze.
 * @param int dsize Velikost pole udržujícího databázi.
 * @return int Indikátor úspěchu (0 = neúspěch , 1 = úspěch).
 *
 * @author \textcopyright{} Jan Naj
 * @version 1.1.4
 */

int dump (entry *data[], int dsize) {

    return 0;
}

```

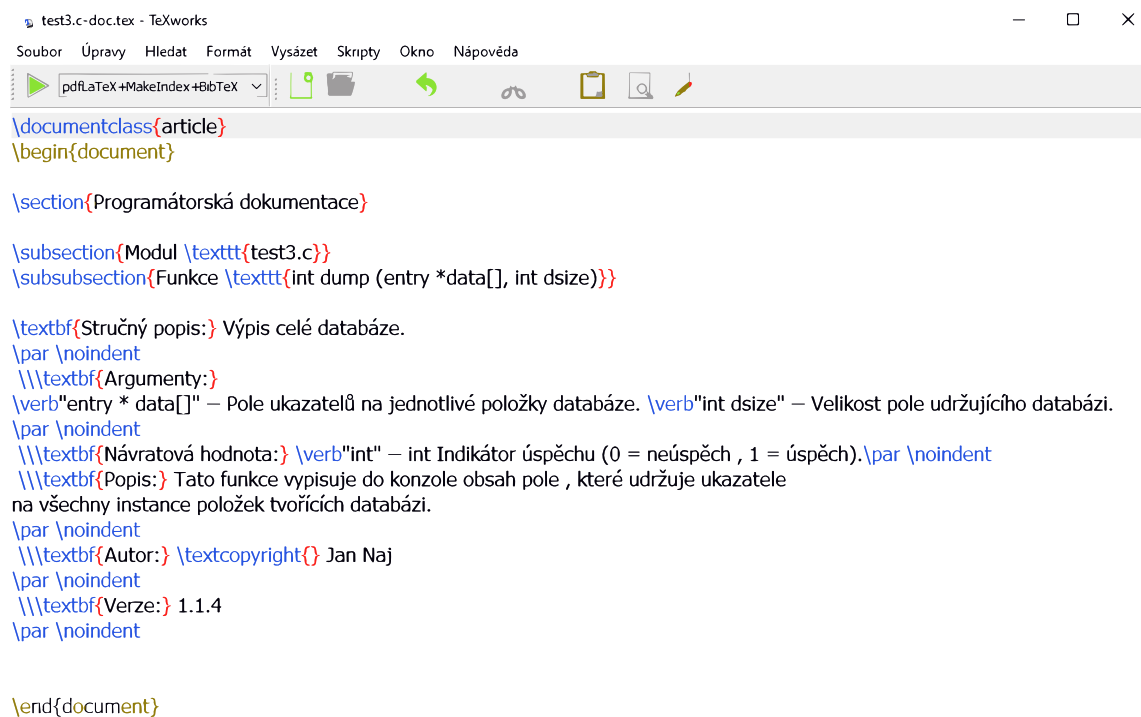
Obrázek 4.2: Vstupní soubor

```

tility.o -Wall -Wextra -pedantic
/mnt/c/Users/Stepan/OneDrive/Plocha/SemestralkaC$ ./ccdoc.exe test3.c
/mnt/c/Users/Stepan/OneDrive/Plocha/SemestralkaC$ _

```

Obrázek 4.3: Spuštění aplikace



```

\documentclass{article}
\begin{document}

\section{Programátorská dokumentace}

\subsection{Modul \texttt{test3.c}}
\subsubsection{Funkce \texttt{int dump (entry *data[], int dsize)}}

\textbf{Stručný popis:} Výpis celé databáze.
\par \noindent
\\ \textbf{Argumenty:}
\verb"entry * data[]" – Pole ukazatelů na jednotlivé položky databáze. \verb"int dsize" – Velikost pole udržujícího databázi.
\par \noindent
\\ \textbf{Návratová hodnota:} \verb"int" – int Indikátor úspěchu (0 = neúspěch , 1 = úspěch).
\par \noindent
\\ \textbf{Popis:} Tato funkce vypisuje do konzole obsah pole , které udržuje ukazatele
na všechny instance položek tvořících databázi.
\par \noindent
\\ \textbf{Autor:} \textcopyright{} Jan Naj
\par \noindent
\\ \textbf{Verze:} 1.1.4
\par \noindent

\end{document}

```

Obrázek 4.4: Výstup v makrojazyce $\text{T}_{\text{E}}\text{X}$

1 Programátorská dokumentace

1.1 Modul test3.c

1.1.1 Funkce `int dump (entry *data[], int dsize)`

Stručný popis: Vypis celé databáze.

Argumenty: `entry * data[]` – Pole ukazatelů na jednotlivé položky databáze.
`int dsize` – Velikost pole udržujícího databázi.

Návratová hodnota: `int` – `int` Indikátor úspěchu (0 = neúspěch , 1 = úspěch).

Popis: Tato funkce vypisuje do konzole obsah pole , které udržuje ukazatele na všechny instance položek tvořících databázi.

Autor: © Jan Naj

Verze: 1.1.4

Obrázek 4.5: Výstup v souboru PDF

4.3 Ukázky výstupu programu

V případě spuštění aplikace bez vstupních parametrů dojde k terminaci aplikace s výstupem chybového hlášení, poukazujícího na absenci vstupního souboru (znázorněno na obrázku 4.6). Aplikace následně vypíše stručný návod k použití v angličtině. Aby bylo možné tento problém vyřešit, je nezbytné zadat alespoň jeden relevantní parametr.

```
slukes@DESKTOP-VF8VP0I:/mnt/c/Users/Stepan/OneDrive/Plocha/SemestralkaC$ ./ccdoc.exe
Error: No input file provided
Enter an input file (obligatory) and an output file (optional)
For instance: test.c |or| test.c output.tex
slukes@DESKTOP-VF8VP0I:/mnt/c/Users/Stepan/OneDrive/Plocha/SemestralkaC$ _
```

Obrázek 4.6: Výstup programu bez zadání parametrů

Při spuštění aplikace s nesprávným prvním, povinným, parametrem aplikace

ukončí provoz a zobrazí chybovou zprávu (viz obrázek 4.7). Tato zpráva upozorňuje buď na neexistenci specifikovaného vstupního souboru nebo na nemožnost jeho otevření. V takovém případě je nutné ověřit, zda daný vstupní soubor skutečně existuje, případně zda se skutečně jedná o soubor se zdrojovým kódem v programovacím jazyce ANSI C.

```
slukes@DESKTOP-VF8VP0I:/mnt/c/Users/Stepan/OneDrive/Plocha/Semestralka$ ./ccdoc.exe nesmysl
Error: File nesmysl does not exist or cannot be opened
slukes@DESKTOP-VF8VP0I:/mnt/c/Users/Stepan/OneDrive/Plocha/Semestralka$ _
```

Obrázek 4.7: Výstup programu s chybným parametrem

Pokud je aplikace spuštěna s více než dvěma parametry, dojde k vyvolání chybové zprávy signalizující nesprávný počet parametrů, jak je ilustrováno na obrázku 4.8. Pro správnou funkčnost je nutné aplikaci spustit s maximálně dvěma relevantními parametry.

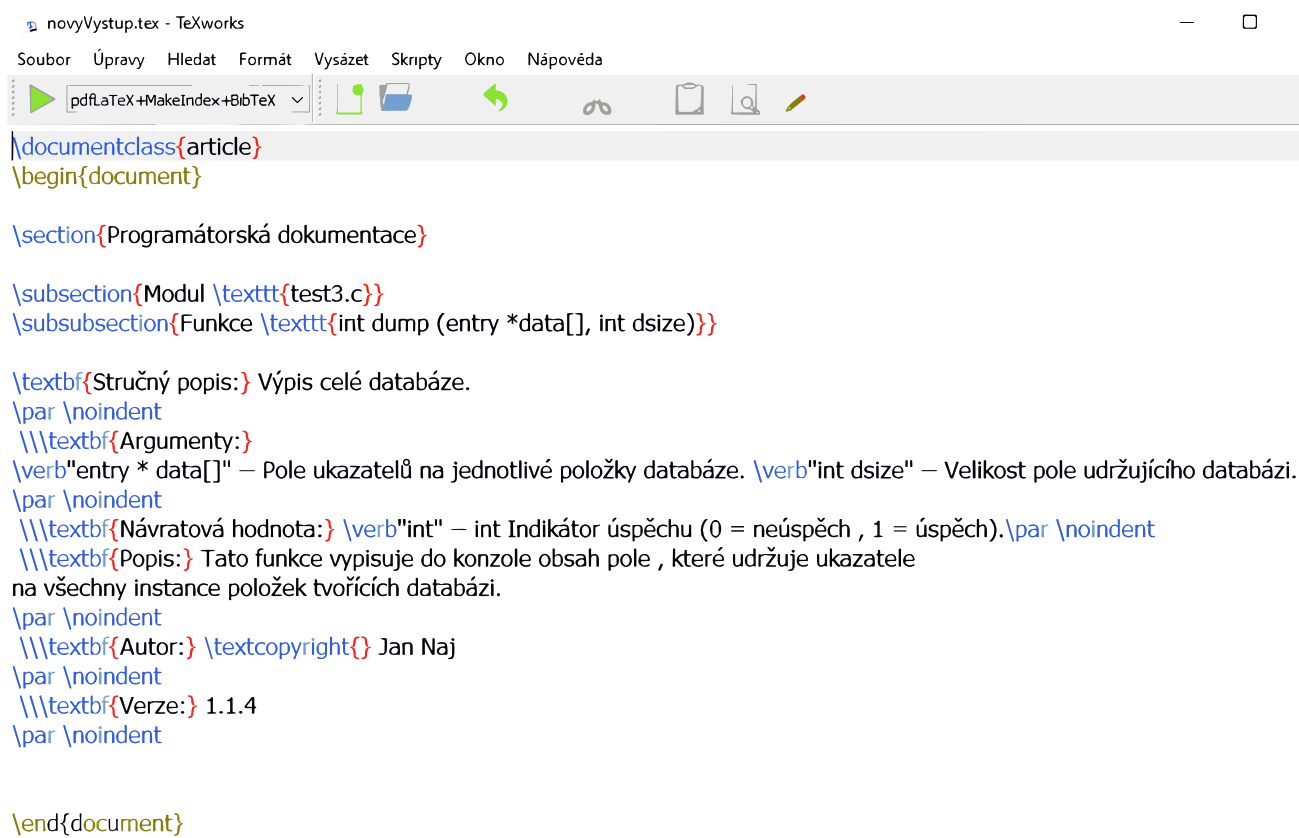
```
slukes@DESKTOP-VF8VP0I:/mnt/c/Users/Stepan/OneDrive/Plocha/Semestralka$ ./ccdoc.exe test3.c vystup vystup
Error: Incorrect number of arguments
slukes@DESKTOP-VF8VP0I:/mnt/c/Users/Stepan/OneDrive/Plocha/Semestralka$ _
```

Obrázek 4.8: Výstup programu s více než dvěma parametry

Při spuštění aplikace s právě dvěma parametry nedochází k výpisu chybové zprávy, což je demonstrováno na obrázku 4.9. V tomto případě bude název vstupního souboru odpovídat jménu zadanému jako druhý parametr, jak je vidět na obrázku 4.10.

```
slukes@DESKTOP-VF8VP0I:/mnt/c/Users/Stepan/OneDrive/Plocha/Semestralka$ ./ccdoc.exe test3.c novyVystup.tex
slukes@DESKTOP-VF8VP0I:/mnt/c/Users/Stepan/OneDrive/Plocha/Semestralka$ _
```

Obrázek 4.9: Výstup programu s dvěma parametry



Obrázek 4.10: Výstup v makrojazyce \TeX

5 Závěr

V závěrečném shrnutí dokumentace lze uvést, že projekt byl úspěšně dokončen s výsledky, které odpovídají požadavkům zadání. Aplikace pro generování dokumentace zdrojového kódu v ANSI C do formátu \LaTeX byla vyvinuta s důrazem na efektivitu, modularitu a rozšiřitelnost. Při zpracování zdrojových souborů dosahuje aplikace vysoké rychlosti, obvykle maximálně do tří vteřin, což výrazně zvyšuje produktivitu uživatele. Během vývoje byly identifikovány a úspěšně řešeny různé technické výzvy, což vedlo k posílení celkové robustnosti a spolehlivosti aplikace.

Pro budoucí vylepšení se nabízejí možnosti jako rozšířená analýza kódu, lepší zpracování složitějších struktur a integrace s dalšími nástroji a platformami. Toto vylepšení by mohlo zahrnovat pokročilejší zpracování komentářů, podporu pro další programovací jazyky nebo integraci s existujícími vývojovými prostředími. Dále by bylo vhodné zaměřit se na optimalizaci uživatelského rozhraní pro zjednodušení procesu generování dokumentace.

Celkově lze konstatovat, že projekt úspěšně splnil zadání a poskytl užitečný nástroj pro generování dokumentace, který může být v budoucnu dále rozšířen a vylepšen.

6 Zdroje

Zdroj Obrázku 2.1 – <https://www.alphacodingskills.com/ds/notes/linked-list.php>

Zdroj Obrázku 2.2 – <https://medium.com/swlh/stacks-and-queues-simplified-ef0f838fc534>