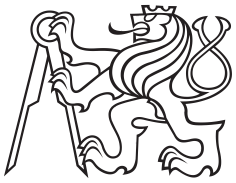
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Kontrola stylu zápisu programu

David Rapant

Vedoucí: RNDr. Ingrid Nagyová, Ph.D.
Studijní program: Otevřená informatika
Specializace: Základy umělé inteligence a počítačových věd
Květen 2023

Poděkování

Rád bych vyjádřil své poděkování své vedoucí práce RNDr. Ingrid Nagyové, Ph.D. za její vedení a podporu během celé práce. Její trpělivost a pochopení bylo pro úspěšné dokončení této práce zásadní.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 26. května 2023

Abstrakt

Tato práce se zaměřuje na analýzu pravidel kódovacího stylu v jazyce C a následné vytvoření nástroje pro kontrolu čitelnosti a udržitelnosti kódu napsaného začátečníky.

Základním cílem je poskytnout nástroj, který bude upozorňovat na chyby a nekonzistence ve zdrojovém kódu. Práce zahrnuje rešerši existujících pravidel a nástrojů pro kontrolu kódovacího stylu, návrh a implementaci nového systému, následované testováním na studentských kódech.

Výsledkem práce je nástroj, který pomůže začínajícím programátorům zlepšit čitelnost a udržitelnost jejich kódu v jazyce C.

Klíčová slova: kódovací styl, pravidla, automatická kontrola, výuka programování, jazyk C

Vedoucí: RNDr. Ingrid Nagyová, Ph.D.

Abstract

This bachelor's thesis focuses on the analysis of coding style rules in the C language with the aim of creating a tool that can automatically check and assess the readability and maintainability of the code created by beginner programmers.

To achieve this, the thesis includes a study of existing rules and tools for checking coding style, based on a new system is designed and implemented. This system is then tested on specific student codes, with the results compared to the results of existing tools.

The outcome of the work is a practical tool, which should help beginners in the C language improve the readability and maintainability of their code.

Keywords: coding style, rules, automatic checking, programming education, C language

Title translation: Checking the Programming Style

Obsah

Obrázky

Tabulky

ctuthesis t1606152353

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Rapant** Jméno: **David** Osobní číslo: **503198**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Specializace: **Základy umělé inteligence a počítačových věd**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Kontrola stylu zápisu programu

Název bakalářské práce anglicky:

Checking the Programming Style

Pokyny pro vypracování:

Cílem práce je analyzovat pravidla pro zápis zdrojového kódu programu v jazyce C a navrhnout a implementovat nástroj, který by dokázal zkontrolovat čitelnost a udržitelnost programu vytvořeného začínajícím programátorem. Účelem nástroje není chyby opravit, ale upozorňovat na ně s úmyslem vést programátora k jejich dodržování. Systém bude prakticky testován na kódech studentů.
1. Proveďte rešerši zdrojů zabývajících se pravidly pro psaní čitelného programového kódu v jazyce C.
2. Seznamte se s chybami ve stylu zápisu programu, které se vyskytují ve studentských programech vytvořených v předmětu Programování v C. Využijte evaluaci cvičících, chyby se pokuste vyhledat i samostatně.
3. Vyhledejte existující nástroje pro kontrolu stylu zápisu programu a analyzujte, zda a jakým způsobem poskytují programátorovi zpětnou vazbu. Pokuste se využít nástroje k hledání chyb ve studentských programech.
4. Analyzujte, která z pravidel a zjištěných chyb studentů lze automaticky (pomocí počítače) vyhodnotit. Navrhněte a implementujte systém, který dokáže zkontrolovat čitelnost a udržitelnost zdrojového kódu zapsaného v jazyce C.
5. Systém otestujte na konkrétních kódech studentů. Srovnejte výsledky vytvořeného nástroje s výsledky z existujících nástrojů a osobní evaluace. Zhodnoťte přínos nástroje.

Seznam doporučené literatury:

[1] Kernighan, Brian W. & Ritchie, Dennis M. „The C Programming Language.“ Prentice-Hall, 1988. Dostupné 31. led. 2023 https://www.cimat.mx/ciencia_para_jovenes/bachillerato/libros/%5BKernighan-Ritchie%5DThe_C_Programming_Language.pdf
[2] Mäkelä, Sami & Leppänen, Ville „Japroch: A Tool for Checking Programming Style“, Finsko: Koli Calling 2004 Conference, 2004. Dostupné 31. led. 2023 http://www.cs.hut.fi/u/archie/KOLI/D12.pdf
[3] „Mistra C“. The MISTRA Consortium Ld. 2021. Dostupné 31. led. 2023 https://www.misra.org.uk/misra-c/
[4] Stallman, R. „Making the best use of C.“ GNU Coding Standards, 2021. Dostupné 31. led. 2023 https://www.gnu.org/prep/standards/html_node/index.html#SEC_Contents
[5] „Linux kernel coding style.“ The kernel development community, 2016. Dostupné 31. led. 2023 https://www.kernel.org/doc/html/v4.10/process/coding-style.html#allocating-memory

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Ingrid Nagyová, Ph.D. kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **13.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

RNDr. Ingrid Nagyová, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Kapitola 1

Úvod

Práce se zaměřuje na analýzu pravidel pro zápis zdrojového kódu programu v jazyce C a návrhem s následnou implementací nástroje, který by dokázal zkontrolovat čitelnost a udržitelnost programu vytvořeného začínajícím programátorem. Hlavní cíl tohoto systému je ulehčit práci vyučujících tím, že sníží čas strávený manuální kontrolou kódu a zvýší efektivitu výuky tím, že poskytne studentům rychlejší zpětnou vazbu o jejich kódovacím stylu.

Pro vyučující programování je stále důležitější zajistit, že studenti nejenže se naučí psát funkční kód, ale také se naučí správným návykům v kódování, jako je dodržování konzistentního a čitelného stylu kódu.

V první části práce se podíváme na samotný kódovací styl, na jeho definici a proč je kódovací styl potřebný a duležitý. Poté se zaměříme na rešerši vybraných zdrojů, které se zabývají pravidly pro psaní čitelného programového kódu v jazyce C. V rámci rešerše shrneme společné aspekty a odlišnosti mezi těmito zdroji, což nám poskytne širší pohled na problematiku kódovacího stylu a jeho význam v různých kontextech.

První polovina druhé části práce se zaměřuje na analýzu chyb ve stylu zápisu programu, které se objevují ve studentských programech vytvořených v předmětu Programování v jazyce C. Druhá polovina se zaobírá analýzou chyb, které lze automaticky vyhodnotit z pravidel a zjištěných chyb studentů.

Ve třetí části práce se zaměříme na zařazení pravidel kódovacího stylu do tabulek, které jsou získané z chyb identifikovaných v předchozí kapitole

společně s pravidly získanými z dalších zdrojů.

Ve čtvrté části jsou zkoumány existující nástroje pro kontrolu stylu zápisu programu a způsob, jakým poskytují programátorovi zpětnou vazbu. Tato analýza pomůže určit nedostatky a výhody současných nástrojů a poskytne základ pro návrh nového nástroje.

V páté a šesté části práce na základě předchozích analýz a poznatků z existujících nástrojů je navržen a implementován systém, který dokáže zkontrolovat námi definovaná pravidla kódovacího stylu a zajistit tak čitelnost a udržitelnost zdrojového kódu zapsaného v jazyce C.

V poslední části práce je systém otestován na konkrétních kódech studentů. Výsledky vytvořeného nástroje jsou srovnány s výsledky existujících nástrojů. Tímto způsobem bude možné zhodnotit přínos navrženého nástroje a určit jeho efektivitu ve srovnání s již dostupnými řešeními.

Výsledkem této bakalářské práce bude nástroj, který pomůže začínajícím programátorům v jazyce C zlepšit čitelnost a udržitelnost jejich zdrojového kódu tím, že je upozorní na chyby a nedostatky ve stylu zápisu programu. Tento nástroj by měl přispět k rychlejšímu a efektivnějšímu učení a zlepšení programovacích dovedností studentů.

Kapitola 2

Kódovací styl

2.1 Definice kódovacího stylu

Definice kódovacího stylu může být nejednoznačná, protože existuje mnoho různých interpretací v literatuře a na internetu, které mohou být správné či nesprávné. Abychom získali jasnější pojem o kódovacím stylu, uvedeme několik definic od známých a zkušených programátorů, které jsou uznávány v komunitě. Tím si každý může lépe uvědomit, co kódovací styl skutečně znamená:

Čistý kód je jednoduchý a přímý, čte se jako dobře napsaná próza a nikdy nezakrývá záměr návrháře, ale je plný ostrých abstrakcí a přímočarých řádků ovládání.[6]

Kód by měl být elegantní a efektivní, logika kódu by měla být přímočará, aby bylo snadné přijít na chyby, aby závislosti byly minimální na snadnou údržbu a ošetření chyb kompletní v souladu s formulovanou strategií a výkon blízký optimu, aby nesváděl lidi k zanezáření kódu k bezprior ním optimalizacím. Čistý kód dělá jednu věc dobře.[6]

Čistý kód může číst a vylepšovat i jiný vývojář než jeho původní autor, má smysluplné názvy a poskytuje jeden způsob, nikoliv více způsobů, jak udělat jednu věc. Kód má minimální závislosti, které jsou výslovně definovány a poskytuje minimální API.[6]

2.2 Důležitost kódovacího stylu

Kódovací styl je zásadní součástí vývoje programů, který přináší řadu výhod pro programátory nováčky, zkušené programátory ale i týmy a celé organizace. Jeho důležitost spočívá v několika klíčových hlediskách:[7]

- **Čitelnost:** Kód, který je dobře strukturovaný a formátovaný je snazší číst a pochopit. To usnadňuje práci programátorům, kteří se mohou rychleji zorientovat v kódu a efektivněji řešit problémy či přidávat nové funkce do kódu.
- **Udržitelnost:** Kód, který je napsán s ohledem na kódovací styl, je snazší udržovat a aktualizovat. To zahrnuje vylepšení výkonu, rozšiřování funkcí a opravy chyb.
- **Spolupráce:** V týmech, kde programátoři pracují na stejném kódu, je kódovací styl klíčový pro účinnou spolupráci. Jednotný styl pomáhá předcházet problémům a nepořádkům, které můžou vzniknout například z důvodu různých stylů formátování.
- **Kvalita kódu:** Dodržování kódovacího stylu může pozitivně ovlivnit kvalitu kódu tím, že zlepšuje čitelnost a snižuje možnost výskytu chyb. Díky tomu mohou být výsledné aplikace odolnější a více spolehlivé.

2.3 Rešerše pravidel kódovacího stylu

Kódovací styl je klíčovým prvkem jakékoli projektu a má značný vliv na čitelnost a udržitelnost kódu. Existuje mnoho zdrojů, které specifikují pravidla kódovacího stylu, včetně motivace pro jejich dodržování, konkrétních pravidel pro různé programovací jazyky a analýz, kde se pravidla často porušují. Následující sekce se zaměřuje na několik vybraných zdrojů a způsoby, jak se zabývají problematikou kódovacího stylu.

GNU Coding Standards[12] poskytují soubor pravidel a doporučení pro psaní kódu v rámci GNU systému. Tato pravidla se zaměřují především na jazyk C, ale mohou být aplikována i na jiné jazyky. Zdroj zdůrazňuje význam konzistence a srozumitelnosti kódu, a to nejen ve smyslu jeho formátování, ale také komentářů.

Linux Kernel Coding Style[13] je dalším zdrojem pravidel kódovacího stylu specifických pro jazyk C. Zatímco některá pravidla jsou specifická pro jádro Linuxu, mnohé z nich jsou obecně platné a mohou být použity při psaní jakéhokoli kódu v jazyce C.

MISRA C[9] je sada pravidel pro psaní bezpečného a spolehlivého kódu v jazyce C, která je široce používaná v kritických systémech, jako je automobilový průmysl. Pravidla se zabývají nejen formátováním kódu, ale také bezpečnostními aspekty a spolehlivostí kódu.

The C Programming Language[5] kniha podrobně popisuje syntaxi a vlastnosti jazyka C. Kromě toho obsahuje doporučení pro psaní čistého a efektivního kódu. Text pokrývá širokou škálu témat, včetně proměnných, konstant, funkcí, ukazatelů a struktur. Její obsah je tak rozsáhlý, že pokrývá jak základní, tak pokročilé koncepty jazyka C, což poskytuje ucelený pohled na použití jazyka C.

2.3.1 Shrnutí rešerše

Každý z těchto zdrojů poskytuje unikátní pohled na pravidla kódovacího stylu. Některé, jako GNU Coding Standards a Linux Kernel Coding Style, se zaměřují na konkrétní pravidla a konvence pro psaní kódu. Jiné, jako je MISRA C, se zaměřují na bezpečnost a spolehlivost kódu. Kniha The C Programming Language nabízí hlubší pohled na jazyk C a obsahuje řadu doporučení pro psaní efektivního kódu.

Kapitola 3

Analýzy

V této kapitole se zaměříme na analýzu chyb ve stylu zápisu programu, které se vyskytují ve studentských programech vytvořených v rámci předmětu Programování v C. Cílem analýzy je identifikovat nejčastější chyby, kterých se studenti při psaní kódu dopouštějí. Dále se podíváme na analýzu chyb, které lze automaticky vyhodnotit na základě kritérií, aby bylo možné provést navrh s následnou implementací systému pro kontrolu čitelnosti a udržitelnosti kódu.

3.1 Analýza studentských programů

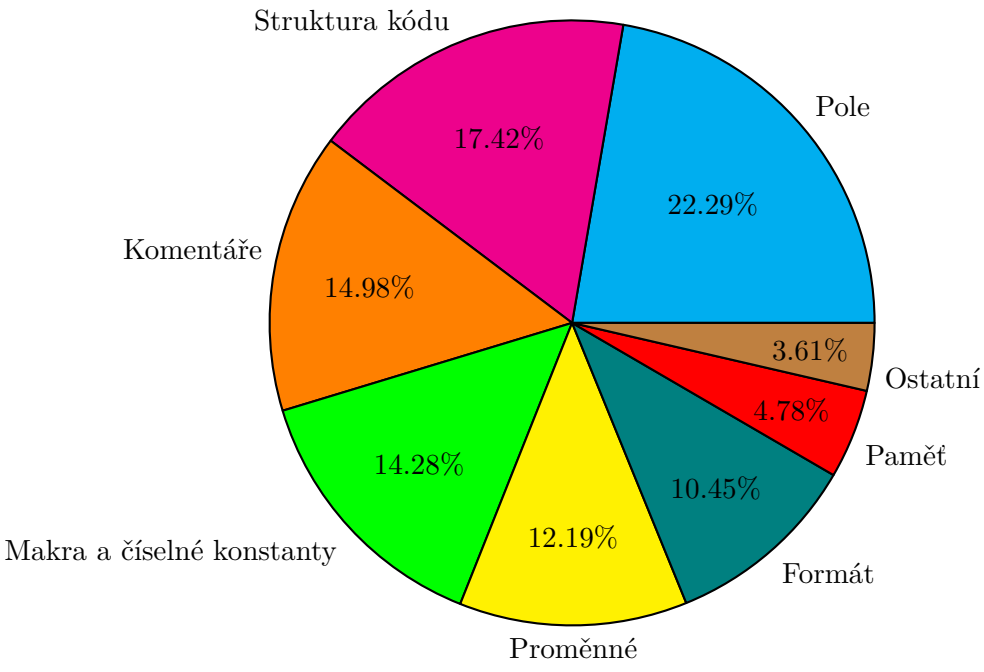
Bylo nám poskytnuto více než 100 hodnocení kódovacího stylu studentských programů, viz. příloha B.2 reviews. Z těchto hodnocení jsme identifikovali celkem 287 případů chyb souvisejících se stylem zápisu kódu.

Tyto chyby nejdříve rozdělíme do kategorií podle jejich charakteristik a povahy, těmi jsou:

- **Pole:** Tato kategorie zahrnuje chyby spojené s použitím globálního pole či špatného indexování.
- **Formát:** Tato kategorie zahrnuje chyby související s uspořádáním kódu, jako jsou odsazení, mezery a poloha závorek.

- **Proměnné:** Do této kategorie patří chyby spojené s pojmenováním proměnných, funkcí a dalších prvků kódu. To může zahrnovat nedostatečně deskriptivní jména nebo jména, která nejsou v souladu s konvencemi pojmenování.
- **Komentáře:** Tato kategorie se týká chyb v komentářích, jako je nedostatek komentářů, absence komentářů u funkcí a styl použití komentářů.
- **Paměť:** Tato kategorie se týká chyb souvisejících s manipulací s pamětí, jako je nesprávné spravování paměti.
- **Makra a číselné konstanty:** Tato kategorie se týká chyb spojených s použitím maker a číselných konstant, jako je například nadměrné používání tzv. *magických čísel* nebo nesprávný styl definice makra.
- **Struktura kódu:** Tato kategorie zahrnuje chyby v celkové struktuře kódu, jako je vysoké zanoření podmínek a cyklů nebo špatné oddělení kódu do funkcí.
- **Ostatní:** Tato kategorie zahrnuje ostatní chyby, které nezapadají do předchozích kategorií, ale stále porušují pravidla kódovacího stylu.

Po rozdělení chyb do kategorií jsme připravili graf 3.1, který názorně ukazuje, jaké procento z celkového počtu chyb představuje každá kategorie a tedy i přehled o tom, které kategorie chyb jsou nejčastější a které naopak méně zastoupené.



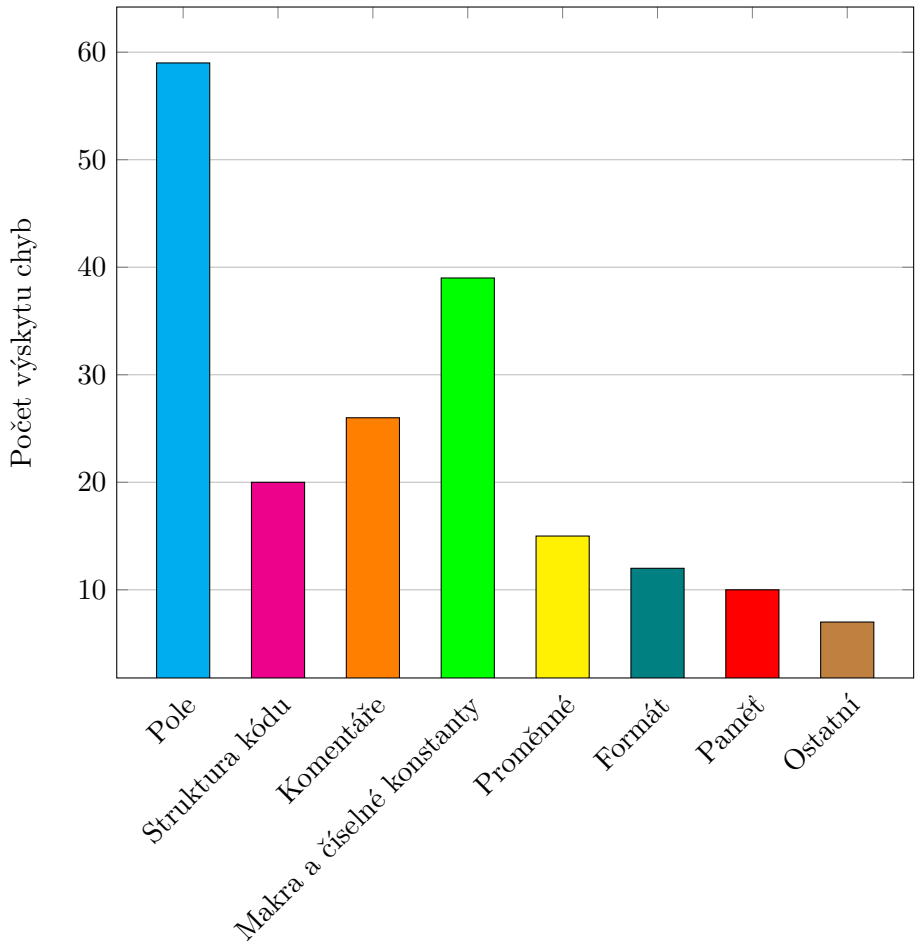
Obrázek 3.1: Procentuální výskyt chyb v kategoriích

Níže uvádíme nejčastější chyby, které se objevily v kódech studentů, pro

každou kategorii.

- **Pole:** Indexování pole pomocí jiného datového typu než unsigned.
- **Struktura kódu:** Opakují se stejné bloky kódu.
- **Komentáře:** Absence komentářů v kódu.
- **Makra a číselné konstanty:** Kód obsahuje magické konstanty.
- **Proměnné:** Proměnné nemají smysluplné názvy.
- **Formát:** Odsazení v kódu je nekonzistentní.
- **Paměť:** V kódu se nekontroluje alokace paměti.
- **Ostatní:** Kontrolova nenulovosti ukazatele není kontrolována s NULL.

Počet těchto jednotlivých chyb z každé kategorie je zobrazen v grafu 3.2.



Obrázek 3.2: Počet výskytu nejčastější chyby v jednotlivých kategorií

3.2 Analýza automaticky vyhodnotitelných chyb

Pro rozhodnutí, jestli je pravidlo automaticky vyhodnotitelné, nám můžou pomoci následující kritéria:

Struktura a jednoznačnost pravidla: Pravidla, která mají jasnou strukturu, jsou většinou jednodušeji automaticky vyhodnotitelná. Pokud tedy pravidlo specifikuje konkrétní formát, je snadnější vytvořit algoritmus, který toto pravidlo zkontroluje.

Komplexita pravidla: Některá pravidla mohou být příliš komplexní nebo závislá na kontextu, což může ztížit jejich automatické vyhodnocení. Například pravidlo o smysluplném pojmenování proměnných by bylo obtížné automaticky vyhodnotit, jelikož pro odhad, zda pojmenování proměnné skutečně odpovídá jejímu účelu, bychom museli nejdříve zjistit smysl kódu. To vyžaduje analýzu kontextu a porozumění programu, což odbočuje od našeho tématu práce.

Samotné vyhodnocení, zda je pravidlo automaticky vyhodnotitelné zobrazíme v následující kapitole do tabulek.

Kapitola 4

Pravidla kódovacího stylu

V této kapitole se zaměříme na rozdělení pravidel kódovacího stylu do tabulek podle již dříve zavedených kategorií. Tato pravidla budou zahrnovat jak chyby identifikované ve studentských programech, které jsou přeformulovány jako pravidla, tak pravidla získaná z literatury a internetových zdrojů.

Pro jednotlivá pravidla evidujeme následující infrmace:

- **Zkratka pravidla:** Zkratka pravidla, pro snažší odkazování na konkrétní pravidla v textu či programové části.
- **Pravidlo:** Samotné pravidlo či doporučení pro kódovací styl.
- **Automatická kontrola:** Ano/Ne - informace, zda je pravidlo vhodné pro automatické vyhodnocení pomocí počítače.
- **Zdroj pravidla:** Označení, zda pravidlo pochází z analýzy studentských programů nebo z literatury či internetových zdrojů. Pravidlo pocházející ze studentských programů bude v tabulkách označen SP, pravidlo pocházející z literatury či internetových zdrojů bude označeno příslušným odkazem na zdroj.
- **Počet výskytů:** Počet případů, kdy byla daná chyba zjištěna v kódech studentů, relevantní tedy pouze pro pravidla získána z kódů studentů.

V tabulkách jsou sloupce zkratka pravidla, automatická kontrola, zdroj pravidla a počet výskytů zkráceny na **ZPR**, **AK**,**ZP** a **PV**, pro lepší rozložení tabulky.

Formát				
ZRP	**Pravidlo**	**AK**	**ZP**	**PV**
F1	Délka řádku je maximálně 80 znaků.	Ano	SP	7
F2	Odsazení je konzistetní v rámci programu.	Ano	SP	12
F3	Před a po tělu funkce je nový řádek.	Ano	SP	1
F4	Formát složenných závorek je sjednocen v rámci programu.	Ano	SP	4
F5	Znaky jiné než ASCII by měly být ojedinělé.	Ano	[3]	N/A
F6	Kód je psaný anglickým jazykem.	Ne	[1]	N/A
F7	Po klíčovém slovu je konzistentní počet mezer.	Ano	SP	2
F8	Mezery jsou konzistentní okolo binárních operátorů.	Ano	SP	4
F9	Mezery jsou konzistentní okolo unárních operátorů.	Ano	[13]	N/A

F7 - Klíčové slova, kterých se pravidlo týká jsou: if, switch, case, for, do, while. F8, F9 - Zdroj pravidel se zabývá specifickým stylem týkajícím se mezer kolem operátorů, avšak naše pravidlo klade důraz na udržování konzistentního stylu v používání mezer. Jak konzistence, tak dodržování osvědčených pravidel jsou důležité, pro začínající programátory budeme dávat přednost konzistenci. Tento přístup bude platit u i některých dalších pravidel.

Tabulka 4.1: Pravidla skupiny formát

Komentáře				
ZRP	**Pravidlo**	**AK**	**ZP**	**PV**
K1	Formát komentu je konzistentní v rámci programu	Ano	[3]	N/A
K2	Před deklarací funkce je komentář.	Ano	SP	17
K3	Program celkově ztrádá na použití komentářů.	Ano	SP	26

K1 - Jedná se o styl, který bude používán buď pro jednořádkové nebo víceřádkové komentáře. Pokud bude daný styl udržován, není důležité, který styl se použije pro jednořádkové nebo víceřádkové komentáře. K3 - Nejlepší situace je, když program nevyžaduje komentáře, protože kód je sám o sobě srozumitelný. Nicméně, v některých případech je třeba komentáře použít, avšak program přeplněný komentáři rovněž není ideální. Ideální situace nastává, když počet komentářů je v rozumném poměru k celkovému počtu řádků kódu.

Tabulka 4.2: Pravidla skupiny komentáře

Paměť				
ZRP	**Pravidlo**	**AK**	**ZP**	**PV**
P1	Dynamická alokace je kontrolovaná.	Ano	SP	10
P2	Alokována pamět je uvolněna.	Ano	SP	4

Tabulka 4.3: Pravidla skupiny paměť

Makra a číselné konstanty				
ZRP	**Pravidlo**	**AK**	**ZP**	**PV**
M1	Pro netriviální numerické literály jsou definovaná makra.	Ano	SP	39
M2	Názvy makra definující konstanty se píší velkými písmeny.	Ano	SP	2

Tabulka 4.4: Pravidla skupiny makra a číselné konstanty

Struktura kódu				
ZRP	**Pravidlo**	**AK**	**ZP**	**PV**
S1	Program je rozdělen do krátkých a jednoduchých funkcí.	Ano	SP	17
S2	Je preferované nízké zanoření podmínek a cyklů.	Ano	SP	8
S3	Program neobsahuje podobné bloky kódu.	Ne	SP	20
S4	V programu se nevyskytují nekonečné cykly.	Ano	SP	5

S4 - Toto pravidlo se týká situace, kdy je ukončovací podmínka umístěna v těle cyklu namísto v jeho hlavičce.

Tabulka 4.5: Pravidla skupiny struktura kódu

Pole				
ZRP	**Pravidlo**	**AK**	**ZP**	**PV**
P1	V programu se nevyskytuje globální pole.	Ano	SP	5
P2	V programu se pro indexování v poli používá unsigned datový typ.	Ano	SP	59

P1 - Samo o sobě není použití globálního pole špatné. Nicméně pro začínající programátory nemusí být globální pole potřebné a může s sebou přinést akorát potenciální problémy.

Tabulka 4.6: Pravidla skupiny pole

Proměnné				
ZRP	**Pravidlo**	**AK**	**ZP**	**PV**
PR1	V programu mají proměnné smysluplné názvy	Ne	SP	15
PR2	V programu jsou proměnné zavedeny co nejblíže k jejich použití.	Ano	SP	6
PR3	V programu jsou názvy proměnných konzistentně ve stejném jazyce.	Ne	SP	6
PR4	V programu jsou názvy proměnných konzistentně ve stejném stylu.	Ano	SP	1
PR5	V programu se nevyskytují globální proměnné.	Ano	SP	7
PR6	Proměnné nemají zbytečně dlouhý název.	Ano	[13]	N/A

PR5 - Stejné jako u pravidla S4.

Tabulka 4.7: Pravidla skupiny proměnné

Ostatní				
ZRP	**Pravidlo**	**AK**	**ZP**	**PV**
O1	Při kontrole nenulovosti ukazatele je třeba provést porovnání se slovem NULL.	Ano	SP	7

Tabulka 4.8: Pravidla skupiny ostatní

4.1 Shrnutí

V této kapitole jsme se zaměřili na identifikaci a třídění pravidel kódovacího stylu, která byla buď identifikována ve studentských programech nebo získána z literatury a internetových zdrojů. Tato pravidla byla rozdělena do osmi kategorií: Formát, Komentáře, Paměť, Makra a číselné konstanty, Struktura kódu, Pole, Proměnné a Ostatní.

Celkově bylo identifikováno a popsání 29 pravidel. Nejvíce pravidel, konkrétně 9, bylo identifikováno v kategorii Formát. Naopak nejméně pravidel, jen jedno, bylo identifikováno v kategorii Ostatní. U 25 pravidel je možná automatická kontrola, u ostatních 4 pravidel není možná automatická kontrola.

Kapitola 5

Současné nástroje

Existuje řada nástrojů určených pro kontrolu stylu kódu a poskytování zpětné vazby programátorům. Následující sekce stručně popisuje některé z těchto nástrojů a analyzuje, jak poskytují zpětnou vazbu. Následně se provede praktické testování těcho nástrojů.

5.1 Představení nástrojů

- **ClangFormat:** Tento nástroj je součástí projektu LLVM a umožňuje automatické formátování kódu v jazycích C, C++, Objective-C a dalších. ClangFormat poskytuje zpětnou vazbu tím, že přeformátuje kód tak, aby splňoval definovaný kódovací styl. [4]

- **Astyle:** Artistic Style, zkráceně AStyle, je nástroj pro formátování kódu, který podporuje řadu programovacích jazyků, včetně C, C++, C# a Java. AStyle je schopen automaticky upravit kód podle zadaných pravidel a preferencí uživatele, což umožňuje snadno a rychle dosáhnout konzistentního stylu kódu napříč celým projektem. [11]

- **GNU Indent:** GNU Indent je nástroj pro úpravu formátu zdrojových kódů v jazyce C. Jeho hlavním účelem je převést zdrojový kód na formát, který je snadněji čitelný a pochopitelný. GNU Indent nabízí širokou škálu nastavitelných parametrů, které umožňují uživatelům přizpůsobit formátování kódu podle svých potřeb a preferencí. [2]

5.2 Praktické testování nástrojů

Nejdříve je třeba vybrat vzorek studentských programů, které by mohly být použity pro testování nástrojů pro kontrolu kódovacího stylu. Z dostupných 20 studentských kódů, viz příloha B.2 student_files byly pro tuto analýzu vybrány tři, každý reprezentující jeden úkol z předmětu Programování v C, které pokrývají různé programovací úlohy a techniky. Tyto programy zahrnují ASCII kreslení domečku, Ceasarovu šifru a Maticové počty.

- **ASCII kreslení domečku:** Tento program vytváří vizuální reprezentaci domečku pomocí znaků ASCII. To zahrnuje použití cyklů, podmínek, výstupních funkcí a manipulaci s řetězci a formátování výstupu.
- **Ceasarova šifra:** Ceasarova šifra je jednoduchá technika šifrování, která posouvá písmena abecedy o určitý počet míst. Tento program používá pole a řetězce, funkce pro manipulaci s řetězci a také využívá ASCII hodnoty znaků.
- **Maticové počty:** Tento program manipuluje s maticemi, používá vnořené cykly pro jejich průchod, provádí nad nimi operace (např. sčítání, násobení), a dynamicky pro ně alokuje paměť.

Po výběru vzorků studentských programů jsme provedli jejich analýzu za použití tří různých nástrojů pro kontrolu formátu a stylu kódu: Clang-Format, AStyle a GNU Indent.

Je důležité poznamenat, že tyto nástroje obecně neupozorňují na chyby v kódu, ale rovnou je opravují. Proto bylo nutné provést analýzu manuálně porovnáváním originálního kódu s kódem opraveným.

Je také důležité zmínit, že tyto nástroje nabízejí řadu konfiguračních možností pro přizpůsobení oprav stylu kódu. Avšak pro naši analýzu jsme všechny tyto nástroje použili s výchozími nastaveními, bez jakékoliv další konfigurace.

Tabulka 5.1 poskytuje detailní přehled chyb, které byly jednotlivými nástroji detekovány v testovaných studentských programech. Cílem této tabulky je poskytnout srovnání schopností jednotlivých nástrojů. Každý řádek v tabulce reprezentuje specifický typ chyby, zatímco sloupce ukazují příslušnou zkratku

daného pravidla z tabulek z kapitoly 4, pokud pro danou chybu existuje příslušné pravidlo, a který nástroj byl schopen danou chybu identifikovat.

Chyby				
Chyba	**clang-format**	**Astyle**	**GNU Indent**	**Kód chyby**
Chybějící mezery okolo binárních a relačních operátorů.	Ano	Ne	Ano	0
Chybějící mezery po čárkách v argumentech funkce.	Ano	Ne	Ano	0
Chybějící mezery po čárkách při definování více proměnných na stejném řádku.	Ano	Ne	Ano	0
Chybějící mezera po středníku ve for cyklu.	Ano	Ano	Ano	0
Chybějící mezera po klíčových slovech (if, for, while).	Ano	Ne	Ano	F7
Nekonzistentní odsazení.	Ano	Ano	Ne	F2
Chybějící mezera mezi "("a "{"pokud jsou na stejné řádce.	Ano	Ano	Ne	0
Chybějící mezera mezi datovým typem a *.	Ano	Ne	Ano	0
Příliš dlouhý řádek	Ano	Ne	Ano	F1
Hvězda indikující ukazatel je připojena k datovému typu, nikoli k názvu ukazatele..	Ano	Ne	Ne	0
Chybějící mezera po čárce v printf.	Ano	Ne	Ano	0
Chybějící mezera mezi else a "{".	Ne	Ano	Ne	0
"{"není na samostatném řádku.	Ano	Ano	Ano	F4
Chybějící mezera mezi jménem/volání funkce a "(".	Ne	Ne	Ano	0
Chybějící mezera mezi free a "(".	Ne	Ne	Ano	0
Chybějící mezery okolo malloc.	Ne	Ne	Ano	0
Chybějící mezera po scanf.	Ne	Ne	Ano	0

Tabulka 5.1: Schopnosti nástrojů detekovat konkrétní typy chyb

V této tabulce je třeba si všimnout, že pravidla detekovaná nástroji pocházejí výhradně z tabulky 4.1, přičemž většina pravidel z kapitoly 4 z našich tabulek nebyly nástroji detekována vůbec.

5.3 Shrnutí nástrojů

AStyle: Nástroj AStyle je méně účinný v detekci chyb než ostatní dva nástroje, ale dokáže odhalit některé chyby, které Clang-Format nebo GNU Indent nedokáží, jako je chybějící mezera mezi "else"a "{". AStyle také dokáže správně detekovat nekonzistentní odsazení a chybějící mezeru mezi "{"a "(", pokud jsou na stejném řádku.

clang-format a GNU Indent: Tyto dva nástroje mají podobnou úroveň účinnosti v detekci chyb. Oba dokážou detekovat širokou škálu chyb, ale existují specifické chyby, které každý z nich detekuje lépe.

5.4 Přínos nástrojů do naší práce

Všechny tři nástroje umožňují určitou úroveň nastavitelnosti. Toto je důležitá vlastnost hned z několika důvodů:

- **Flexibilita pro různé předměty:** Pokud by se tento systém měl uplatnit v jiném programovacím předmětu zaměřeném na jazyk C, mohly by se požadavky na kódovací styl lišit. V takovém kontextu by se nástroj s možností přizpůsobení pravidel stal nezbytným.

  Podpora pro individuální styl: Studenti, kteří již mají svůj vlastní kódovací styl, mohou chtít používat nástroj, který je schopen jejich styl respektovat. Nastavitelný nástroj může být přizpůsoben tak, aby vyhovoval jejich osobním preferencím.

  Podpora pro vyučování: Některé chyby v kódovacím stylu mohou být důležitější pro výuku než jiné. Nastavitelný nástroj umožňuje vyučujícím si vybrat, která pravidla by se měla kontrolovat a která naopak jsou pro daný předmět nepodstatná.

Na základě výše uvedených úvah v naší práci zahrneme podporu nastavitelnosti v podobě konfiguračního souboru.

Kapitola 6

Návrh systému

V této kapitole se budeme zabývat požadavky na náš systém a architekturou našeho systému.

6.1 Specifikace požadavků

Popíšeme požadavky na náš systém pro dva hlavní typy uživatelu, těmi jsou studenti a vyučující.

Specifikace požadavků:

- Systém bude umožňovat studentům nahrání svého kódu pro kontrolu.
- Systém bude umožňovat vyučujícím hromadné nahrání studentských kódů pro kontrolu.
- Systém bude schopen identifikovat chyby a nedostatky v nahraných kódech týkající se kódovacího stylu.
- Systém bude schopen pro studenty nalezené chyby a nedostatky zobratit v přehledné a srozumitelné formě přímo v terminálu.
- Systém bude schopen pro vyučující nalezené chyby a nedostatky v kódech studentů zobratit v přehledné a srozumitelné formě pomocí výstupních souborů.

- Systém bude umožňovat přizbůsobit některá pravidla kódovacího stylu dle potřeb a preferencí pomocí příkazové řádku či konfiguračního souboru.
- Systém bude umožňovat deaktivaci vybraných kontrol pravidel dle potřeby.

6.2 Technologické volby

Jazyk a vývojové prostředí: Pro vývoj systému byl vybrán jazyk Python[14]. Python byl vybrán nejen pro svou čitelnost, stručnost a širokou škálu dostupných knihoven a nástrojů, ale také pro svou podporu v oblasti vývoje nástrojů pro analýzu kódu. Navzdory nižší výkonnosti Pythonu, která může být nevýhodou v některých případech, rychlost zpracování v kontextu našeho systému není primární prioritou, pokud dokáže systém vyhodnotit programy v přijatelném čase. Vývoj bude probíhat v prostředí Visual Studio Code[8], které bylo vybráno pro své pokročilé funkce pro efektivní vývoj.

Parsování: Pro analýzu zdrojového kódu v jazyce C využijeme nástroje ANTLR[10]. ANTLR je nástroj pro generování lexerů a parserů, který je hojně využíván v mnoha oblastech, jako jsou kompilátory, analýza kódu, vývoj jazyků a další. Při návrhu tohoto systému jsme využili ANTLR z následujících důvodů:

- **Flexibilita:** ANTLR umožňuje definovat gramatiku pro téměř jakýkoliv jazyk, což z něj činí velmi flexibilní nástroj. V našem případě nám to umožní analyzovat zdrojový kód v jazyce C a získat z něj potřebné informace pro kontrolu stylu.
- **Jednoduchost použití:** ANTLR je navržen tak, aby byl snadno použitelný. Gramatiky jsou definovány v jasném a srozumitelném formátu, což vede k rychlému navyknutí na tyto gramatiky a ke zjednodušení procesu vývoje.
- **Podpora Pythonu:** ANTLR podporuje generování lexerů a parserů pro Python, což je samozřejmě pro nás důležité, protože náš systém bude napsán v tomto jazyce.
- **Automatická generace stromu parsování:** ANTLR automaticky generuje strom parsování (AST - Abstract Syntax Tree)[15] na základě dané gramatiky. Tento AST pak lze snadno procházet a analyzovat, což je pro naše potřeby zásadní.

Takže ANTLR nám poskytuje silný, flexibilní a dobře podporovaný nástroj pro analýzu zdrojového kódu, což je klíčová část našeho systému. Jeho využití nám umožní soustředit se na samotnou kontrolu stylu, aniž bychom museli řešit složitosti parsování zdrojového kódu. Jedinou potenciální nevýhodou použití ANTLR pro tento projekt je jeho výkonnost a složitost, která může být pro náš konkrétní případ až nadbytečná. Nicméně vzhledem k potenciálním budoucím rozšířením systému nám použití ANTLR poskytne dostatečnou flexibilitu.

6.3 Architektura systému

Systém je navržen jako modulární, což usnadňuje jeho rozšiřování. Skládá se z několika hlavních komponent, které spolu spolupracují:

- **Vstupní modul** - čte vstupní soubory a předává je lexikálnímu analyzátoru.
- **Lexer** - rozdělí vstup na tokeny, které jsou pak předány parseru.
- **Parser** - vytvoří syntaktický strom na základě tokenů.
- **Analyzátor kódovacího stylu** - analyzuje kódovací styl v syntaktickém stromu.
- **Kontrolní modul** - kontroluje kód podle nastavených pravidel a zaznamenává případné chyby.

6.4 Konfigurace pravidel

Uživatelé mají možnost konfigurovat pravidla pro kontrolu kódovacího stylu pomocí konfiguračního souboru. Tento konfigurační soubor umožní uživatelům nastavit například preferovaný styl identifikátorů a další jiné pravidla.

6.5 Spuštění systému

Systém je navržen tak, aby byl co nejsnadněji použitelný. Uživatelé budou moct systém spustit přímo z příkazové řádky a specifikovat vstupní soubor nebo složku a volitelný konfigurační soubor.

6.6 Analýza a kontrola kódovacího stylu

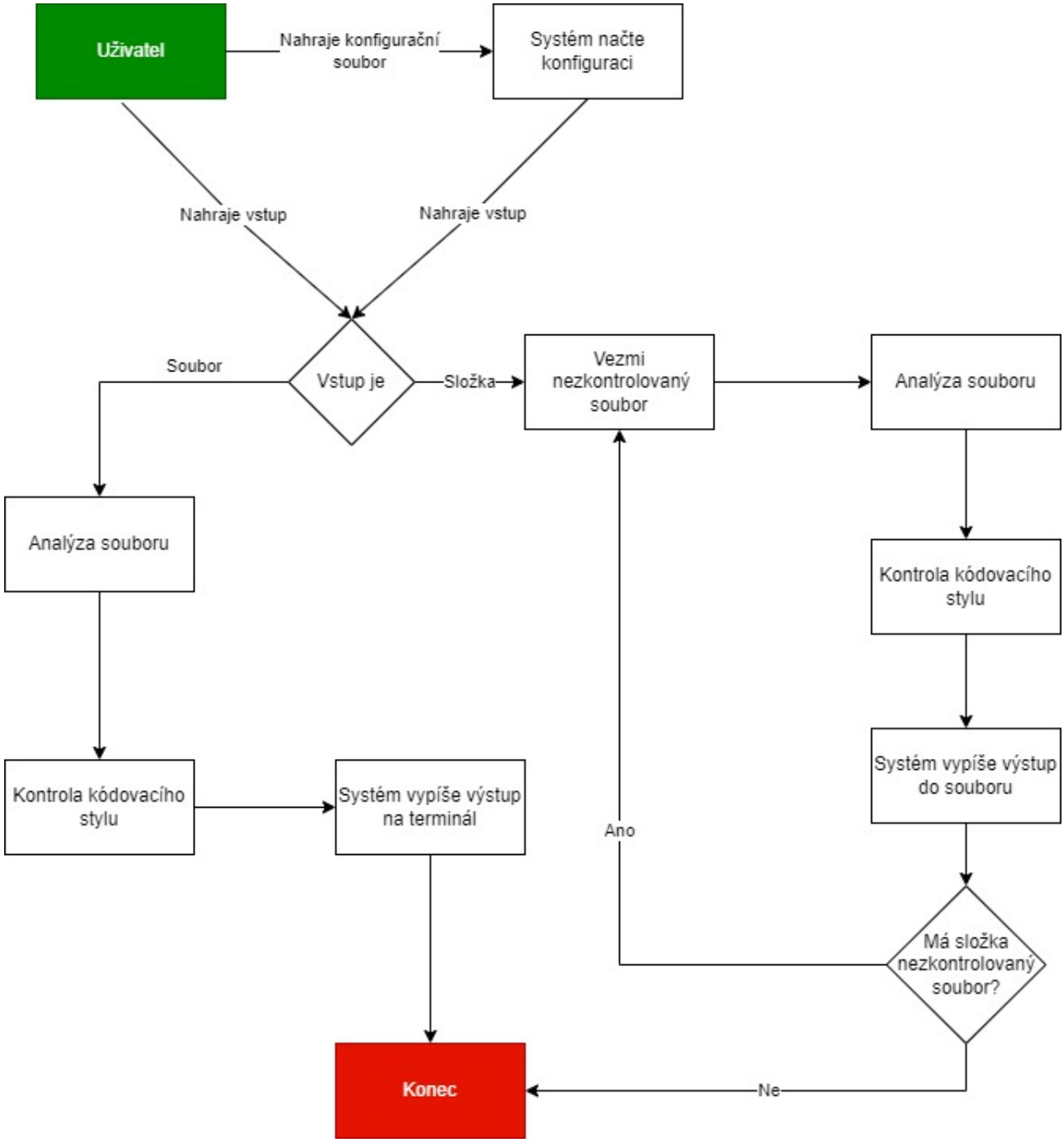
Analýza a kontrola kódovacího stylu představují jádro našeho systému. Analýza nejdříve projde syntaktický strom a rozpoznává charakteristiky kódovacího stylu. Jakmile je analýza dokončena, kontrolní modul projde synktatický strom znova a ověří, zda kód splňuje definovaná pravidla.

6.7 Výstup systému

Po dokončení analýzy a kontroly systém vygeneruje výstup, který obsahuje informace o nalezených chybách a rozpoznaném kódovacím stylu. V případě kontroly více souborů systém vygeneruje samostatný výstupní soubor pro každý zkontrolovaný soubor.

6.8 Diagram chodu programu

Pro lepší představivost, jak systém zpracovává vstup a výstup přikládám níže diagram. Zelená barva představuje počáteční stav a červený koncový stav.



Obrázek 6.1: Diagram chodu programu.

Kapitola 7

Implementace systému

V této kapitole se zaměříme na implementaci našeho systému, včetně procesu zpracování vstupu a detailního popisu, jak systém funguje.

7.1 Vstupní data a spuštění systému

Uživatel může systému předložit program v jazyce C nebo složku obsahující více programů. Toto umožňuje jak jednotlivé studenty, tak vyučující, aby mohli ověřit dodržování kódovacích stylů v jednom či více souborech najednou. Systém také umožňuje předložit konfigurační soubor, kterým může uživatel upravit nebo vypnout kontrolu některých pravidel podle svých potřeb.

Systém se spouští v terminálu pomocí následujícího příkazu:

```
$ python3 main.py input --config config_file
```

Tady main.py je hlavní modul systému, input je vstupní soubor nebo složka a –config je přepínač, který za něj dovolí na vstup přidat konfigurační soubor config_file.

7.2 Zpracování a parsování vstupu

Po spuštění začne systém zpracovávat vstupní soubor či soubory. Nejdříve je vstup pročten pomocí třídy InputStream z knihovny ANTLR, která umožňuje číst data ze vstupního proudu. Následně je tento proud rozdělen na jednotlivé tokeny pomocí lexikálního analyzátoru (CLexer). Tokeny představují základní jednotky, které parser rozpozná. Parser (CParser) pak vezme proud těchto tokenů a zahájí proces parsování, přičemž vytváří syntaktický strom (parse tree), který reprezentuje strukturu zdrojového kódu. Parser ke správnému rozparsování tokenů potřebuje mít definovanou gramatiku jazyka, který je analyzován. V našem případě jsme využili existující gramatiku pro jazyk C poskytnutou ANTLR, kterou jsme dle potřeby upravili a přizpůsobili našim požadavkům. Následně je vytvořený strom procházen pomocí instance ParseTreeWalker, která volá metody třídy StyleAnalyzer na každém uzlu stromu.

7.3 Analýza kódovacího stylu

Z vytvořeného syntaktického stromu, dále už jen AST, je poté analyzován kódovací styl. Zjišťuje se například preferovaný styl identifikátorů a složenných závorek.

7.4 Kontrola kódovacího stylu

Jakmile je provedena analýza a případná konfigurace pravidel, systém začne kontrolovat kódovací stylu. CStyleChecker, naše třída na kontrolu kódovacího stylu začne procházet AST, počínaje kořenem a postupuje rekurzivně směrem dolů. Pro každý uzel AST je volána odpovídající pravidlová metoda, která vyhodnocuje dané pravidlo kódovacího stylu a pokud je to potřeba, pokračuje dále v procházení AST.

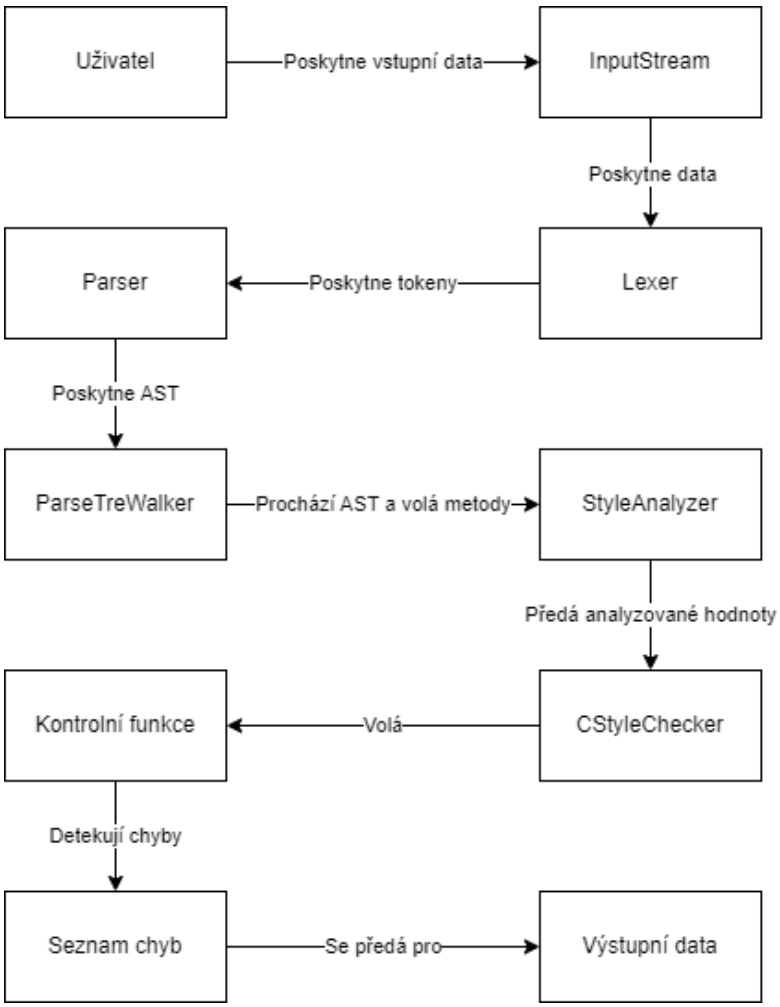
Toto procházení AST je realizováno pomocí metod enterRule a exitRule, enterRule a exitRule jsou metody, které jsou součástí ANTLR rozhraní. Toto rozhraní je generováno ANTLR pro každou gramatiku, kterou vytvoříme. enterRule a exitRule metody jsou volány, když parser vstoupí/vystoupí do/z určitého pravidla v gramatice. Tato metoda může být přepsána, aby provedla

určité akce při vstupu/výstupu do/z pravidla. Například pro uzel reprezentující definici funkce existuje metoda enterFunctionDefinition, která je zavolána při vstupu do daného uzlu a exitFunctionDefinition, která je zavolána při odchodu z uzlu. V našem systému jsou většinou využívány metody typu enterRule, protože jsou pro naše účely více vhodné. Tyto metody poté volají při průchodu AST námi implementované funkce pro kontrolu konkrétních pravidel kódovacího stylu. Níže uvádíme příklady těchto funkcí a jejich použití.

- **check_function_description:** Kontroluje zda má funkce nad sebou komentář.
- **check_curly_braces:** Kontroluje zda složenné závorky funkce jsou ve správném stylu.
- **check_spaces_between_fun:** Kontroluje zda mají funkce mezi sebou určitý počet volných řádku.
- **check_function_length:** Kontroluje zda funkce není příliš dlouhá.
- **get_indent_sizes:** Tato funkce sama o sobě nic nekontroluje, pouze zapisuje velikost mezery od začátků řádku do prvního znaku na stejné řádce pro pozdější kontrolu.

Podobným způsobem jsou napsány i ostatní kontrolující funkce, které se volají uvnitř enterRule a exitRule metod. Pokud některá z těchto kontrolních funkcí detekuje chybu, zapíše se tato chyba do seznamu chyb, který je na konci celé kontroly vypsán.

Pro lepší představu jak systém provádí celý proces popsaný v textu výše přikládáme diagram, který tento proces může přiblížit.



Obrázek 7.1: Proces systému.

7.4.1 Kontrolovaná pravidla

Naše implementace systému pro kontrolu kódovacího stylu zahrnuje seznam pravidel uvedené v příloze C.

7.5 Výstup systému

Po dokončení procházení celého AST a jeho kontroly systém vypíše informace o kódovacím stylu, který byl v daném souboru nastaven nebo analyzován.

Dále systém vypíše na terminál nalezené chyby, pokud byl nahrán pouze jeden soubor.

V případě, že byla pro kontrolu nahrána celá složka souborů, systém vloží nalezené chyby do samostatného souboru pro každý zkontrolovaný soubor. Tyto soubory budou mít názvy odpovídající jménům původních souborů a budou uloženy do speciální výstupní složky. Toto uspořádání umožňuje efektivní zpracování výsledků kontroly kódovacího stylu pro velký počet souborů.

Tímto způsobem systém zajistí komplexní a přizpůsobitelnou kontrolu kódovacího stylu v C pro jednotlivé soubory nebo celé sady souborů. Nabízí tak uživatelům efektivní nástroj pro kontrolu a zlepšení kvality kódu, ať už pro jednotlivé programátory, nebo pro výukové účely.

7.6 Unit testy

Pro ověření správnosti našeho systému jsme implementovali sadu unit testů. Tyto testy ověřují, že funkce pro detekci chyb správně rozpoznají možné chyby v kódu a že celý systém správně zpracovává vstupní soubory a generuje správné výstupní soubory.

Kapitola 8

Testování systému

V této kapitole otestujeme náš systém na zdrojových kódech studentů oproti již současným nástrojům z kapitoly 5.

V tabulce 8.1 představujeme detekční schopnosti tří existujících nástrojů a našeho vlastního nástroje na zjišťování chyb v kódovacím stylu v různých souborech. Pro náš nástroj je počet chyb snadno dostupný, neboť stačí spočítat počet chyb vypsaných ve výstupu našeho systému. U ostatních tří nástrojů jsme museli přistoupit k improvizaci, neboť tyto nástroje neposkytují přímý náhled na detekované chyby a místo toho soubory přímo opraví. Tím pádem jsme byli nuceni srovnávat počet změn v originálním souboru s již opraveným souborem, toho jsme dosáhli použití skriptu z přílohy B.3 counter_script.sh, který vyhodnotí každý soubor těmito nástroji a následně porovná odlišnost řádků. Je proto třeba brát počet detekovaných chyb prvních třech nástrojů s jistou rezervou.

Pro každý kontrolovaný soubor v tabulce je nástroj, který detekoval nejvíce chyb, označen zelenou barvou. Naopak nástroj, který identifikoval nejmenší počet chyb, je označen červenou barvou.

Chyby				
Soubor	**clang-format**	**Astyle**	**GNU Indent**	**Náš systém**
main1.c	114	20	134	14
main2.c	131	13	137	19
main3.c	268	248	302	70
main4.c	241	103	252	118
main5.c	173	167	193	221
main6.c	171	69	175	78
main7.c	268	67	295	103
main8.c	82	46	96	70
main9.c	186	22	196	782
main10.c	112	41	111	66
main11.c	119	209	165	217
main12.c	76	97	103	76
main13.c	177	53	174	50
main14.c	171	13	147	28
main15.c	181	193	181	104
main16.c	195	93	214	283
main17.c	85	24	92	88
main18.c	150	109	197	141
main19.c	98	51	118	75
main20.c	173	108	199	47

Tabulka 8.1: Srovnání počtu detekovaných chyb

Na základě výsledků představených v tabulce 8.1 můžeme vyvodit několik závěrů o schopnostech našeho systému detekovat chyby v kódovacím stylu.

Jedním z hlavních poznatků je, že nástroj GNU Indent je celkově účinnější ve srovnání s ostatními nástroji, zatímco nástroj Astyle se ukázal jako nejméně účinný. Toto je založeno na výchozím nastavení všech nástrojů. Je důležité si uvědomit, že všechny tyto nástroje poskytují množství nastavení, která lze přizpůsobit konkrétnímu programu. Pokud by byly tyto nástroje upraveny pro určitý styl kódu, mohly by výsledky vypadat značně odlišně.

Oproti tomu, naše testy ukázaly, že našemu systému se daří přizpůsobit stylu kódu, pokud je tento styl konzistentní. Tento faktor zvyšuje univerzálnost a flexibilitu našeho systému. I když se našemu systému nepodařilo získat mnoho ani červených, ani zelených označení, což by se dalo interpretovat tak, že se pohybuje někde v průměru mezi ostatními nástroji, je důležité podotknout, že naše hlavní cílová skupina nejsou extrémní případy, ale spíše běžné situace, kde je styl kódu konzistentní. Je důležité poznamenat a zohlednit, že náš systém nekontroluje tolik pravidel, jako ostatní zmíněné nástroje.

Kapitola 9

Závěr

V průběhu této práce jsme se detailně seznámili s konceptem stylu kódu a jeho významem pro programování. Studovali jsme existující nástroje pro analýzu stylu kódu a vyvinuli jsme vlastní systém, který dokáže detekovat chyby v kódovacím stylu a přizpůsobit se specifickému stylu kódu.

Výsledky našeho systému ukazují, že je schopen se téměř srovnat s existujícími nástroji. Kromě toho, naše analýza ukázala, že náš systém má schopnost se přizpůsobit specifickému stylu kódu, což je významná výhoda v porovnání oproti existujícím nástrojům, které často nerespektují osobní preference programátora.

Přestože výsledky jsou pozitivní, jsou zde stále oblasti, kde je možné naši práci dále rozvíjet a vylepšovat. Zahrnuje to například zlepšení detekčních schopností našeho systému a přidání více pravidel, která system bude kontrolovat.

Celkově je možné konstatovat, že náš systém představuje významný krok vpřed v oblasti detekce chyb v kódovacím stylu. Jeho schopnost se přizpůsobit a respektovat konkrétní styl kódu nabízí nové možnosti jak pro studenty programování, tak i pro vyučující. Studenti mohou využít systému k zlepšení svých dovedností a dodržování kódovacích standardů, zatímco vyučující mohou systém využít jako nástroj pro poskytování zpětné vazby a hodnocení práce studentů. Tímto způsobem systém přináší výhody pro oba tyto uživatelské skupiny a přispívá k lepšímu výsledku výuky programování v jazyce C.

Příloha A

Literatura

[1] J. Faigl. *Pravidla pro psaní čitelného kódu*. https://cw.fel.cvut.cz/b201/courses/b0b36prp/resources/tessun/start, 2020.

[2] GNU Project. *GNU Indent*. https://www.gnu.org/software/indent/, 2023.

[3] Google. *Google C++ Style Guide*. https://google.github.io/styleguide/cppguide.html, 2008.

[4] D. Jasper. *ClangFormat*. https://clang.llvm.org/docs/ClangFormat.html, 2013.

[5] Kernighan, B. W., Ritchie, and D. M. *The C Programming Language*. https://www.cimat.mx/ciencia_para_jovenes/bachillerato/libros/%5BKernighan-Ritchie%5DThe_C_Programming_Language.pdf, 1988.

[6] R. C. Martinl. *A Handbook of Agile Software Craftsmanship*. 2008.

[7] S. C. McConnell. *Code Complete*. http://aroma.vn/web/wp-content/uploads/2016/11/code-complete-2nd-edition-v413hav.pdf, 2004.

[8] Microsoft. *Visual Studio Code*. https://www.antlr.org/, 2015.

[9] MISRA. *Guidelines for the use of the C language in critical systems*. https://electrovolt.ir/wp-content/uploads/2022/09/MISRA-C_2012_-Guidelines-for-the-Use-of-the-C-Language-in-Critical-Systems-Motor-Industry-Resear pdf, 2013.

[10] T. Parr. *ANTLR*. https://www.antlr.org/, 1992.

[11] J. Pattee and A. Simon. *Artistic Style*. https://code.visualstudio.com/, 2015.

[12] R. Stallman. *GNU Coding Standards*. https://www.gnu.org/prep/standards/standards.pdf, 2021.

[13] The kernel development community. *Linux kernel coding style*. https://www.kernel.org/doc/html/v4.10/process/coding-style.html, 2016.

[14] G. van Rossum. *Python*. https://www.python.org/, 1991.

[15] Wikipedia contributors. *Abstract syntax tree*. https://en.wikipedia.org/wiki/Abstract_syntax_tree.

Příloha B

Externí přílohy

cstylechecker.zip: Kompletní zdrojový kód systému včetně ostatních podpůrných souborů využité k vyvynutí našeho systému.

B.1 Zdrojový kód

- **main.py:** Hlavní modul spouštející zpracování vstupu, analýzu kódu, opravu kódu s následným vypsáním výstupu.
- **process_input.py:** Modul zpracující vstup do systému společně s konfiguračním souborem.
- **analyzer.py:** Modul analyzující kódovací styl vstupních souborů.
- **style_checker.py:** Modul volající kontrolní funkce z control_modules.
- **control_modules:** Moduly implementující samotné kontroly kódovacího stylu
- **parser_files:** Složka obsahující vygenerované ANTLR pro správné fungování tohoto nástroje včetně lexeru (CLexer.py), parseru (CParser.py) a gramatiky (C.g4).
- **unit_files:** Složka obsahující unit testy pro náš systém.
- **unit_tests:** Složka obsahující C kódy použité jako vstup do našich unit testů.
- **config.ini:** Ukázkový konfigurační soubor.

B.2 Studentské soubory:

- **student_files:** Složka obsahující C kódy, které byly využity pro závěrečné testování nástrojů v kapitole 8 a v praktickém testování v kapitole 5.2.
- **reviews:** Složka obsahující hodnocení kódovacího stylu studentských programů.

B.3 Ostatní:

- **counter_script.sh:** Skript použitý pro spočítání chyb nalezených existujícími nástroji v kapitole 8.
- **tests:** Námi vytvořené C kódy používané během implementace systému pro testování.
- **errors.ods:** Tabulka spočtených chyb ke každé kategorii ze složky reviews.

Příloha C

Kontrolovaná pravidla

Kontrolovaná pravidla	
ZRP	**Pravidlo**
F1	Délka řádku je maximálně 80 znaků.
F2	Odsazení je konzistetní v rámci programu.
F3	Před a po tělu funkce je nový řádek.
F4	Formát složenných závorek je sjednocen v rámci programu.
F7	Po klíčovém slovu je konzistentní počet mezer.
F8	Mezery jsou konzistentní okolo binárních operátorů.
F9	Mezery jsou konzistentní okolo unárních operátorů.
S1	Program je rozdělen do krátkých a jednoduchých funkcí.
S2	Je preferované nízké zanoření podmínek a cyklů.
PR4	V programu jsou názvy proměnných konzistentně ve stejném stylu.
PR6	Proměnné nemají zbytečně dlouhý název.
M1	Pro netriviální numerické literály jsou definovaná makra.
M2	Názvy makra definující konstanty se píší velkými písmeny.
K1	Formát komentu je konzistentní v rámci programu
K2	Před deklarací funkce je komentář.