Teorie kompilátorů

Petr Horáček

Obsah

1 Úvod

Kompilátor je počítačový program, který překládá program napsaný v jednom jazyce na program napsaný v jiném jazyce. To znamená, že kompilátory potřebují znát gramatiku a kontext jak vstupního jazyka tak výstupního jazyka. Struktura kompálátoru pak vychází z tohoto jednoduchého pozorování. Kompilátor jakožto program se skládá z front-end části, která má za úkol zpracovat vstupní jazyk a back-end, který má za úkol pracovat s výstupním jazykem. Front-end a back-end jsou propojeny pomocí vrstvy middle-end která je pracuje s tzv. mezikódem IR.

Vedle kompilátorů se pak využívají také interpretery, které se od kompilátorů liší tím, že místo vytvoření spustitelného souboru s přeloženým programem daný program okamžitě vykonávají. Takovému interpreteru se v praxi často říká virtuální stroj.

Kompilátory a interpretery spolu úzce souvisejí a jsou často zaměňovány. Součástí interpreteru je často kompilátor, který program ve zdrojovém jazyce překládá nejčastěji do podoby **bytecodu**, který umožňuje rychleji vykonávat výsledný program. Jiný způsob je překlad bytecodu do strojového kódu lokálního procesoru a bez uložení do souboru jej vykonáva. To je známé jako **Just-In-Time** kompilace nebo také **JIT**.

Počítačový program je jednoduchá sekvence abstraktních operací zapsaná v **Programovacím jazyce** - formální jazyk, navržený pro vyjádření výpočtů.

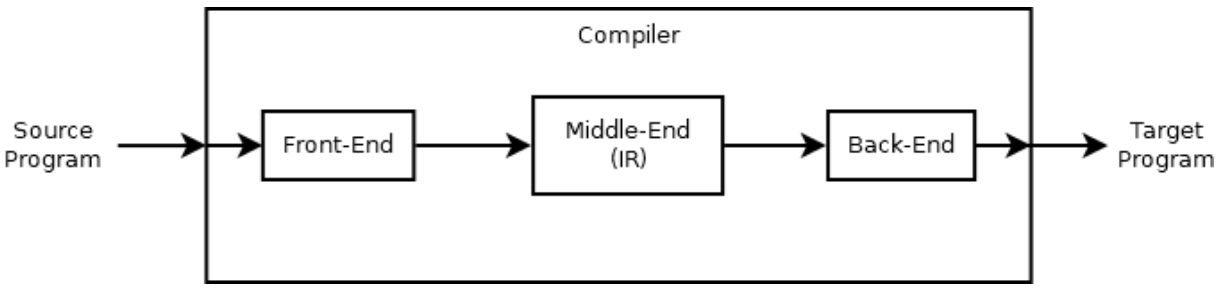
2 Struktura kompilátoru

Moderní kompilátory jsou navrženy tak, aby byly flexibilní a modulární. Tato flexibilita umožňuje kompilátoru efektivně zpracovávat různé programovací jazyky (na vstupu) a generovat kód pro různé platformy (na výstupu). Klíčem k této modularitě je standardizovaná intermediate representation (IR), kterou využívá střední část architektury kompilátoru - tzv. middle-end.

Kompilátor se nejčastěji skládá ze tří základních logických částí:

- **Front-End** - Zpracovává vstupní programový kód.
- **Middle-End** - mechanismy, které převádí abstraktní syntaktický strom na mezikód IR, případně provádějí některé platformě nezávislé optimalizace. Dále se provádí alokace paměti a registrů a jiných zdrojů procesoru.
- **Back-End** - definice cílové platformy. Na této úrovni se provádějí platformně závislé optimalizce a převádí se IR na instrukce procesoru

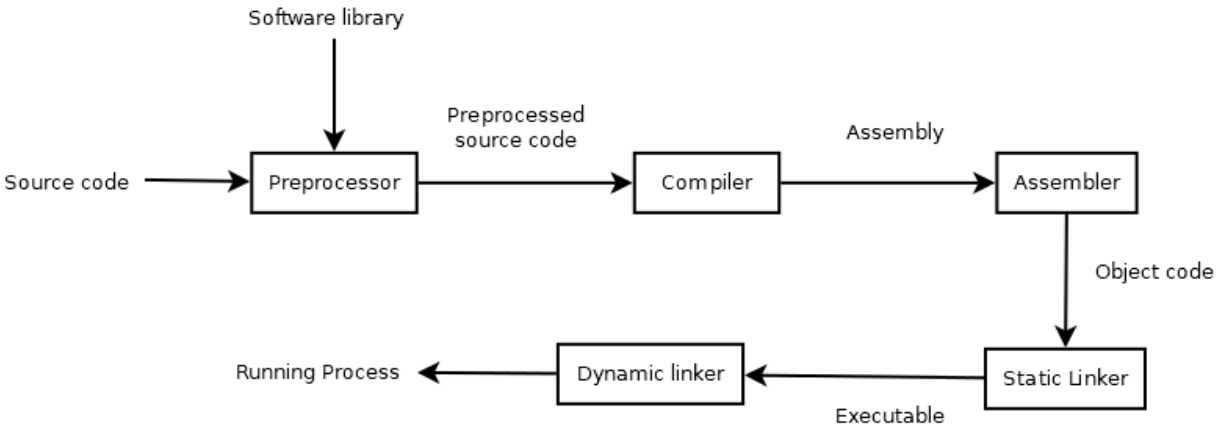
Výhodou tohoto členění architektury komilátoru je, že v případě, že je standardizovaný formát IR na vrstvě **Middle-End** je možné dynamicky měnit části **Front-End** pro zpracovávání zcela jiného programovacího jazyka a nebo **Back-End** pro generování strojových instrukcí pro zcela jinou platformu se zachování stejné gramatiky programovacího jazyka. Změna cílového jazyka, který kompilátor generuje na svém výstupu se nazývá **Compiler Retargeting**



Obrázek 2.0.1 Struktura kompilátoru

2.1 Fáze překladu

Kompilátor je jednou z komponent nástrojové sady **(toolchain)** používané pro vytvření spustitelného souboru z programu ve formě textu ve zdrojovém jazyce. Typicky při spuštění jediného příkazu k překladu je vyvolána celá sekvence prorgamů na pozadí.



Obrázek 2.1.1 Proces překladu zdrojových kódů

2.1.1 Preprocesing

Preprocesor je nástroj v procesu překladu zdrojového kódu, jehož účelem je provést úpravy kódu před samotnou kompilací. Preprocesor zpracovává specifické příkazy nebo direktivy, které mohou zahrnovat vkládání externích souborů, definici konstant a maker, podmíněné sestavení kódu nebo jeho úpravy ještě před analýzou syntaxe. Tímto způsobem může vývojář upravovat kód dynamicky podle prostředí, zvyšovat přehlednost pomocí symbolických konstant, či minimalizovat opakování kódu s použitím maker. Výstupem preprocesingu je čistý kód, připravený pro kompilaci, který splňuje požadavky a strukturu definovanou direktivami preprocesoru. Preprocesor samotný ale není nezbytnou součástí kompilátorů, proto nemusí být ve všech případe v kompilátorech přítomné.

2.1.2 Kompilace

Fáze kompilace je klíčovým krokem v procesu překladu zdrojového kódu na spustitelný kód. Během této fáze kompilátor analyzuje syntaxi a sémantiku kódu, převádí ho z původního programovacího jazyka do střední reprezentace (IR) a následně aplikuje optimalizace, které zlepší efektivitu výsledného programu. Poté kompilátor IR transformuje na instrukce konkrétní strojové architektury nebo na assemblerový kód, který je blízký instrukční sadě procesoru. Účelem fáze kompilace je tedy vytvořit nízkoúrovňový kód, který je přesně přizpůsoben cílové platformě, přičemž zachovává logiku a funkčnost původního zdrojového kódu.

2.1.3 Assembler

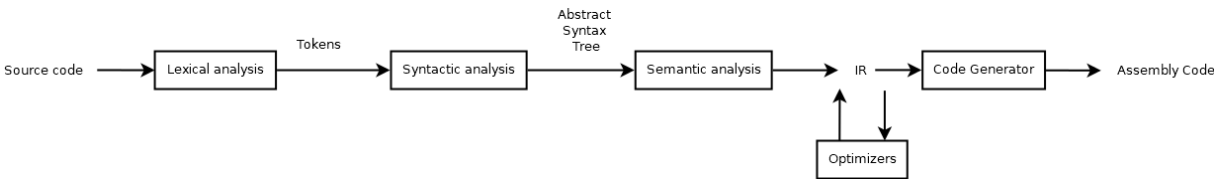
Fáze překladu assemblerového kódu do objektového kódu slouží k převodu nízkoúrovňových instrukcí, napsaných v assembleru, na strojový kód, který může být přímo interpretován procesorem. V této fázi assembler převádí každou instrukci na konkrétní binární reprezentaci odpovídající instrukční sadě cílového procesoru. Kromě toho vytváří tabulky symbolů a další metadata, která pomáhají při pozdějším sestavení (linkování). Výstupem této fáze je objektový kód, který není zatím kompletním spustitelným programem, ale obsahuje všechny základní instrukce a data, které budou spojeny a finalizovány ve fázi linkování.

2.1.4 Linkování

Fáze linkování je posledním krokem v překladu programu, jehož účelem je spojit jednotlivé objektové soubory a knihovny do jednoho kompletního spustitelného souboru. Linker v této fázi vyhledá a propojí všechny odkazy na symboly (například funkce a globální proměnné) mezi různými částmi programu a knihovnami. Rovněž přiřadí výsledné adresy paměti pro tyto symboly, čímž zajistí, že programové instrukce a data jsou správně propojené a připravené k běhu. Výsledkem je hotový, spustitelný program, který může operační systém načíst a provést.

2.2 Fáze kompilace

Fáze kompilace je klíčovou součástí procesu překladu, ve kterém je zdrojový kód napsaný v programovacím jazyce převáděn do podoby, kterou lze efektivně provést na cílové platformě. Tato fáze zahrnuje několik kroků, které postupně transformují zdrojový kód do podoby, která je blízká strojovému kódu, a přitom optimalizuje výkon výsledného programu. V procesu kompilace lze rozlišit několik základních kroků, lexikální analýza, syntaktická analýza, sémantická analýza, generování mezikódu (IR), optimalizace, generování kódu cílové platformny.



Obrázek 2.2.1 Proces překladu zdrojových kódů

2.2.1 Lexikální analýza

Lexikální analýza je prvním krokem ve fázi kompilace, kde kompilátor rozkládá zdrojový kód na základní jednotky zvané **tokeny**. Tokeny jsou sekvence znaků, které tvoří logické stavební bloky kódu, jako jsou klíčová slova, identifikátory, operátory a literály. Lexikální analyzátor prochází zdrojový kód, identifikuje tyto tokeny a zpracovává je tak, aby byly připraveny pro další fázi. Pokud zjistí chyby, například neznámé znaky, kompilace je přerušena a uživatel je informován.

2.2.2 Syntaktická analýza

Během syntaktické analýzy kompilátor vytváří strukturu programu, která odpovídá gramatice daného jazyka. Syntaktický analyzátor používá tokeny generované lexikálním analyzátorem k vytvoření **syntaktického stromu (AST)**, který zobrazuje hierarchickou strukturu kódu. Tento strom odhaluje vztahy mezi jednotlivými konstrukcemi jazyka, jako jsou výrazy, příkazy a bloky kódu. Tato fáze zajišťuje, že program odpovídá pravidlům jazyka a že má správnou syntaxi.

2.2.3 Sémantická analýza

V sémantické analýze kompilátor kontroluje, zda syntaktický strom kódu dodržuje významová pravidla jazyka, tedy že je program logicky správný. Během této fáze se provádí například kontrola typů proměnných, deklarací a definic, volání funkcí nebo kompatibility operandů v operacích. Výsledkem je zajištění, že program má platný význam a bude fungovat správně v kontextu daného jazyka. Sémantická analýza také rozšiřuje syntaktický strom o doplňující informace, které jsou později využívány při optimalizaci.

2.2.4 Optimalizace

Optimalizace je klíčovou fází, která umožňuje kompilátoru zlepšit výkonnost a efektivitu výsledného kódu. Optimalizační algoritmy se aplikují na střední reprezentaci a zajišťují například minimalizaci počtu instrukcí, efektivní využití paměti nebo redukci redundancí. Existují různé úrovně optimalizací - od základních, jako je eliminace zbytečných výpočtů, až po pokročilé techniky, jako je paralelizace nebo inlining funkcí. Optimalizace se snaží nalézt rovnováhu mezi výkonem programu a jeho velikostí.

2.2.5 Generování cílvého kódu

V závěrečné fázi kompilace je optimalizovaná střední reprezentace převedena na cílový kód specifický pro platformu, pro kterou je program určen. Kompilátor převádí IR na instrukce v assembleru, nebo přímo na strojový kód, přičemž zohledňuje konkrétní instrukční sadu procesoru cílové architektury. Výstupem této fáze je sada instrukcí, které jsou interpretovatelné procesorem a zajišťují požadovanou funkčnost programu.

3 Lexikální analýza

3.1 Preprocessing

4 Syntaktická analýza

5 Zachytávání chyb

Chyba v softwaru je nesprávné nebo neočekávané chování programu nebo systému, které se vyskytuje při jeho spuštění, používání nebo vývoji. Chyby mohou mít různé příčiny, od problémů v kódu až po nesprávné požadavky od uživatele. Mohou být objeveny během vývoje, testování nebo při ostrém používání aplikace.

Chyby se obvykle dělí do několika kategorií:

1. **Kompilační chyby** - chyby, které vznikly chybou ve zdrojových kódech, nebo chybným nastavením překladu
2. **Logické chyby** - Chyby, kdy program běží bez výjimek, ale neprovádí správně požadovanou funkci, což vede k nesprávným výsledkům.
3. **Chyby běhu** - Chyby, které se objeví během vykonávání programu, například dělení nulou, přístup k neexistujícímu souboru nebo pokus o přístup k nealokované paměti.
4. **Chyby návrhu** - Chyby, které vznikají na úrovni architektury nebo návrhu softwarového systému a mohou mít dlouhodobý vliv na jeho funkčnost a údržbu.
5. **Chyby uživatele** - Chyby způsobené nesprávným použitím programu nebo nedorozuměním mezi uživatelem a systémem.

Chyby, které mohou nastat ve fázi kompilace, se pak dělí na:

1. **Lexikální chyby** - Chyby vzniklé na úrovni slovní reprezentace zdrojového kódu, například chyba v zápisu desetinného čísla, identifikátoru nebo klíčového slova
2. **Syntaktické chyby** - Chyby, které vznikají při porušení pravidel syntaxe programovacího jazyka. Tyto chyby jsou obvykle odhaleny během fáze kompilace nebo interpretace kódu.
3. **Sémantické chyby** - Chyby, které vznikají ve chvíly kdy je zdrojový kód syntakticky správny, ale kód přesto není platný, například nesprávné použití operací na datový typ, nebo změna hodnoty konstantní proměnné
4. **Logické chyby** - Chyby, které nastanou ve chyvíly kdy je kód ve všech směrech platný, ale přesto jeho použití nedává smysl, například nekonečné smyčky, mrtvý kód, ...

Varování v softwaru jsou upozornění na potenciální problémy nebo neoptimální chování programu, které nezpůsobují okamžité selhání nebo ne-

správné výsledky, ale mohou vést k chybám nebo neefektivnímu fungování v budoucnu. Na rozdíl od chyb, které obvykle zastavují běh programu nebo způsobují nesprávné chování, varování upozorňují vývojáře nebo uživatele na situace, které by si zasloužily pozornost, ale nebrání v pokračování programu.

5.1 Zachytávání chyb na úrovni kompilátoru

Úlohou zachytávání chyb je detekovat chyby, které mohou nastat v různých fázích překladu a předat tuto informaci uživatelovi, který tak může podniknout příslušné kroky k nápravě. Vztah mezi chybami a varováními spočívá v tom, jakým způsobem kompilátor reaguje na různé typy problémů, které se vyskytují v kódu během procesu překladu. Tyto dvě kategorie odrážejí rozdílné úrovně závažnosti problémů a určují, jakým způsobem kompilátor zachází s jejich detekováním a hlášením.

5.2 Metody zachytávání chyb

Kompilátory používají různé metody pro zachycení chyb a varování, a to včetně přístupu k jejich následnému ošetření.

5.2.1 Immediate Error Reporting (Okamžité hlášení chyb)

Tento přístup znamená, že kompilátor zastaví zpracování kódu a vypíše chybu ihned v místě jejího vzniku. Tato metoda je běžná při zachytávání syntaxových chyb. Výhodou okamžitého hlášení je rychlá identifikace chyby, nevýhodou může být časté přerušování procesu překladu, což může být neefektivní, pokud má kód více chyb.

5.2.2 Error Buffering (Ukládání chyb do bufferu)

V tomto přístupu kompilátor shromažďuje chyby v chybovém bufferu (nebo logu), aby mohl detekovat více chyb najednou, než přeruší kompilaci. Tento přístup je užitečný, protože vývojáři poskytuje přehled všech chyb v kódu, což umožňuje efektivnější opravu více problémů najednou. Po dokončení fáze překladu se z bufferu vypíše seznam všech chyb, což zjednodušuje ladění a údržbu kódu.

5.2.3 Recovery Methods (Metody zotavení)

Kompilátory často implementují metody pro zotavení po detekci chyby, což jim umožňuje pokračovat v překladu a najít další chyby. Metody pro zotavení jsou často kombinovány s metodou ukládání chyb do bufferu. Mezi tyto metody patří:

- **Panic Mode**: Kompilátor po zjištění chyby ignoruje zbytek aktuálního bloku nebo příkazu a pokračuje až od dalšího vhodného bodu, jako je konec funkce nebo příkaz. Tento přístup je rychlý, ale může vynechat některé chyby v rámci ignorovaného úseku.
- **Phrase-Level Recovery**: Při tomto přístupu se kompilátor pokusí upravit nebo doplnit kód (např. doplněním chybějícího středníku) a pokračovat v překladu, což umožňuje identifikaci dalších chyb. Tento přístup je však složitější a nemusí vždy vést k správnému zpracování kódu.
- **Error Productions**: Jedná se o speciální případ metody Phrase-Level Recovery. Kompilátor je naprogramován tak, aby rozeznal běžné chyby a pokusil se je automaticky opravit. Například, pokud chybí závorka v podmíněném výrazu, kompilátor ji může doplnit, aby umožnil pokračování v překladu. Tato metoda však může být náročná na implementaci a vyžaduje znalost běžných chybových vzorců.

5.3 Definice chyb a varování v kompilátoru

Definice chyb a varování v kompilátoru závisí na metodě, která jse použita k jejich zachytávání. V případě metody Imediate Error Reporting jsou veškeré informace předány na vstup

6 Modulový systém

Modulový systém v programovacích jazycích je mechanismus, který slouží k organizaci a strukturování kódu do logických částí zvaných moduly. Modulový systém umožňuje programátorům rozdělit rozsáhlý kód do menších, lépe spravovatelných jednotek a zajišťuje kontrolu nad viditelností a přístupností symbolů (jako jsou funkce, proměnné, třídy) mezi těmito jednotkami. To poskytuje několik klíčových výhod:

Modulový systém určuje, jakým způsobem jsou organizovány a spravovány různé části kódu. To ovlivňuje nejen syntax a sémantiku jazyka, ale i to, jak překladač zpracovává různé části kódu.

7 Sémantická analýza

8 Intermediate reprezentace (IR)

9 Optimalizace

10 Správa paměti a běhové prostředí

11 Alokace registrů procesoru

12 Generování kódu

13 Just-In-Time (JIT) kompilace

14 Linkování a načítání