PROJEKT PŘEKLADAČE

pro předměty Formální jazyky a překladače a Algoritmy

Tým 097, varianta I

Tom Barbořík (xbarbo06) – 25%
Pavel Kaleta (xkalet05) – 25%
Miroslav Tichavský (xticha04) – 25%
David Myška (xmyska05) – 25%

Rozšíření: projekt neimplementuje žádné z možných rozšíření

Lexikální analyzátor

Syntaktický analyzátor (bez zpracování výrazů)

Pracoval: Miroslav Tichavský, Tom Barbořík, Pavel Kaleta

Pro implementaci syntaktické analýzy jsme zvolili doporučenou metodu **rekurzivního sestupu**. Největším úskalím, před vlastní implementací, bylo správné zvolení pravidel pro konstrukci bezkontextové gramatiky, pomocí které jsme vytvořili LL – tabulku (viz příloha), tak aby odpovídala zadaným konstrukcím jazyka IFJ17.

Vstup:

Syntaktický analyzátor volá scanner (lexikální analyzátor), který v globální proměnné předkládá typ načteného tokenu a případně další potřebné detaily o tokenu.

Výstup:

Výstupem analýzy je vyhodnocení, zda je zdrojový kód zapsán syntakticky správně nebo došlo k syntaktické chybě. Dále dochází k naplnění tabulky symbolů a kontrole sémantiky. Také nezajišťuje vytvoření syntaktických stromů, které jsou následně předkládány generátoru cílového kódu.

Implementace:

Pro každý neterminál gramatiky je implementovaná funkce, která pomocí kontroly příchozích tokenů vyhodnocuje, zda tokeny korespondují s gramatickými pravidly pro daný neterminál (viz LL – tabulka). Neterminály jsou zástupné symboly, které se během rekurzivního sestupu snažíme v několika krocích nahradit sekvencí terminálů.

Během rekurzivního volání funkcí dochází také ke kontrole sémantické správnosti zdrojového kódu, k tomu je využita tabulka symbolů implementována v modulu **symtable.c**. Do tabulky vkládáme identifikátory proměnných a funkcí a při jejich volání kontrolujeme, jestli došlo ke správné definici či deklaraci. Kontrolujeme také, jestli nedochází k redefinici proměnných a funkcí.

Vykonáváním rekurzivního sestupu je zároveň prováděno plnění syntaktických stromů, které se následně předloží generátoru cílového kódu. Pro plnění jsou využity funkce implementovány v modulu **syntaxtree.c**.

Tabulka symbolů

Pracoval: David Myška

Tabulka symbolů je implementována pomocí binárního vyhledávacího stromu. Vyhledávání probíhá podle klíče odpovídajícímu názvu proměnné nebo funkce. Každý uzel si s sebou nese několik informací, typ uzlu, zda se jedná o funkci nebo proměnnou, datový typ, pro proměnné označuje její datový typ a pro funkce návratový typ. Dále pak seznam parametrů, který je definovaný pouze pro funkce.

Syntaktický strom

Pracoval: David Myška

Syntaktický strom jsme použili jako rozhraní mezi syntaktickou analýzou a generátorem výstupního kódu. Generátoru je pak předáván celý seznam stromů, implementovaný pomocí dynamického pole, obsahující vnitřní reprezentaci celého vstupního kódu.

Jednotlivé stromy, popřípadě jeho uzly, se vytvářejí pomocí skupiny funkcí "*`STcreate`*...", například funkce *`STcreateIntConst(int value)`* vytvoří uzel pro celočíselnou konstantu odpovídající hodnotě zadané jejím parametrem. Některé funkce mají jako parametr podstrom vytvořené také těmito funkcemi, proto jsou funkce "*`STcreate`*..." psány tak, aby je bylo možné zanořovat, například voláním "*`STcreateExpr(STcreateDoubleConst(1.75), "*", STcreateVar("var", d_double))`*" se vytvoří strom odpovídající výrazu "*1.75 * var*". Každá z funkcí přijímající nějaký podstrom jako parametr přísně kontroluje, zda zadaný parametr má povolený typ, například do podmínky nelze vložit výpis na obrazovku. Pro přidaní stromu do seznamu pak jen stačí zavolat funkci *`Ladd(s_list *list, s_stree tree)`*, popřípadě jako parametr "*tree*" lze opět zadat přímo příkazy ze skupiny "*`STcreate`*...".