Relatório do projeto de Compiladores 2023/24

Compilador para a linguagem UC

André Rodrigues Costa Pinto 2021213497

Pedro Tiago Gomes Ramalho 2019248594

Secção i) Gramática reescrita

Para reescrever a gramática baseamo-nos na ficha p3_syntatic_analysis que realizamos nas aulas pl. Começamos por reescrever a Gramática inicial em notação EBNF para uma notação que fosse reconhecida pelo yacc. Ao compilar o executavel verificamos que existiam várias ambiguidades na gramática, que fomos resolvendo alterando a forma como esta foi construida. Para perceber que excertos originavam as ambiguidades utilizamos a flag -v nesta linha do ficheiro build.sh: 'yacc -d -v -t -g -Wcounterexamples –report=all uccompiler.y lex uccompiler.l'. As alterações relativamente à gramática EBNF fornecida foram:

Expr: Foram adicionadas regras específicas para cada operador, eliminando a ambiguidade e garantindo uma ordem específica de avaliação, além disso as regras foram ajustadas para melhorar a consistência e clareza da gramática.

FunctionBody: Foi modificado para permitir a presença opcional de `DeclarationsAndStatements`.

Adições de Tokens, Prioridades e Associatividades: Adicionamos tokens para operadores e outros símbolos relevantes. Foram definidas as prioridades e associatividades para operadores, garantindo uma análise correta.

Secção ii) Algoritmos e estruturas de dados da AST e da tabela de símbolos:

Para desenvolver estes algoritmos baseamo-nos nas fichas p4_abstract_syntax e na ficha p5_semantic_analysis, assim como nos algoritmos do compilador Petit que se encontra disponivel no github do professor Raul.

Ficheiros "ast.h" e "ast.c":

A estrutura base é o nó (`struct node`), que contém informações essenciais sobre os elementos da AST. Cada nó possui uma categoria (`enum category`), que identifica o papel do nó na gramática do programa. Além disso, um nó possui um token, que é o valor associado ao nó.

Cada nó pode ter filhos, que são armazenados numa lista ligada (`struct node_list`). A lista de nós representa os filhos de um nó específico. Cada elemento da lista contém um ponteiro para um nó e um ponteiro para o próximo elemento da lista.

Além disso, cada nó pode ter um ponteiro para seu nó pai (`struct node *parent`). Isso permite navegar de um nó para seu pai, facilitando operações que envolvem a análise ascendente na árvore.

Funções de inserção na àrvore: `newnode`: Cria um novo nó com a categoria e token passados por parâmetro. `addchild`: Adiciona um nó filho a um nó pai. `addbrother`: Adiciona um novo nó irmão ao final da lista de irmãos de um nó.

Funções de apresentação da àrvore: `show`: Exibe a AST de forma hierárquica, indicando a categoria e o token de cada nó. `show_all`: Similar a `show`, mas acrescenta à frente dos nós currespondentes a expressões, o tipo da variavel a ele atribuida

Ficheiros "semantics.h" e "semantics.c":

Estruturas: `struct symbol_list`: Representa uma lista de símbolos, contendo informações como identificador, tipo, nó associado e um ponteiro para o próximo símbolo na lista. `struct symbol_table_list`: Representa uma lista de tabelas de símbolos, contendo o nome da tabela, a tabela em si e um ponteiro para a próxima tabela.

Funções: `int check_program(struct node *program)`: Função principal para a análise semântica do programa. Cria as tabelas globais e das funções, verifica declarações e definições de funções, e retorna o número de erros semânticos encontrados. `struct symbol_list *insert_symbol(struct symbol_list *table, char *identifier, enum type type, struct node *node)`: Insere um novo símbolo numa tabela, a menos que o símbolo já esteja presente. Retorna o novo símbolo inserido ou NULL se o símbolo já existir. `struct symbol_list *search_symbol(struct symbol_list *table, char *identifier)`: Procura um símbolo por identificador em uma tabela e retorna o símbolo encontrado ou NULL se não existir. `struct symbol_table_list *search_table(struct symbol_table_list *tables, char *identifier)`: Procura uma tabela de símbolos por identificador e retorna a tabela encontrada ou NULL se não existir.

`void show_symbol_table()`: Apresenta as tabelas de símbolos global e de funções, mostrando identificadores, tipos e, quando aplicável, informações adicionais.

Secção iii) Geração de código

Não realizamos a meta 4 do projeto.