Universidade Federal do Amazonas
Instituto de Computação
Compiladores
Prof. Edson Nascimento Silva Júnior

Trabalho 03 – Geração de Código Intermediário

Descrição geral

O objetivo deste trabalho consiste na implementação de um compilador funcional, para a linguagem que chamaremos a partir de agora de first2023.. Esta etapa do trabalho consiste na geração de código intermediário a partir da AST (Árvore Sintática Abstrata).

Tarefas necessárias:

A geração de código deve fazer as seguintes tarefas:

A. Implementação da estrutura de TACs (Códigos de instrução de três endereços) – Um novo módulo deve prover uma estrutura com código de instrução e três ponteiros para a tabela de símbolos, permitindo que essas instruções sejam encadeadas em listas de instruções. Cada instrução indica a execução de uma operação com zero, um ou dois operandos que serão valores em memória, e cujo resultado também está em memória, sendo os endereços de memória representados pelos símbolos na tabela de símbolos. O módulo deve prover rotinas utilitárias para criar, imprimir e unir listas de instruções;
B. Criação de símbolos temporários e labels – Devem ser feitas duas rotinas auxiliares que criam novos símbolos na tabela de símbolos (hash), uma para variáveis temporárias e outra para labels. Elas serão usadas na geração de código para guardar sub--resultados de cada operação e para marcar os pontos de desvio no fluxo de execução;
C. Geração de código – faça uma rotina que percorre a AST recursivamente, retornando um trecho de código intermediário (lista de TACs) para cada nodo visitado. Essa rotina primeiro processa os nodos filhos, armazena os trechos de código gerados para cada um deles, depois testa o nodo atual e gera o código correspondente para este nodo. A geração em geral consiste na criação de uma ou mais novas instruções (TACs), união dos trechos das sub--árvores e dessas novas instruções, opcionalmente com a criação de novos símbolos intermediários e labels, retornando um trecho de código completo deste novo nodo;

Desenvolvimento

As instruções em código intermediário servem para isolar as tarefas de geração da sequência básica de instruções dos detalhes e formatos específicos de uma arquitetura-alvo.

Além disso, a geração usada neste trabalho emprega técnicas genéricas de forma funcional, didática;

porém, pode ser otimizada de várias formas antes da geração de código assembly. Dois exemplos de otimização são a reutilização de símbolos temporários em expressões e o uso de registradores. Entretanto, essas otimizações não fazem parte desta etapa do trabalho.

A geração de código será feita de baixo para cima e da esquerda para a direita, na árvore. O modo mais simples de encadear novas instruções é representar os trechos de código como listas encadeadas invertidas (stack), isto é, com um ponteiro para a última instrução de um trecho, e cada instrução apontando para a anterior. Ao final da geração, escreva uma rotina que percorre o código completo e inverta a lista de forma que se possa escrever o código na ordem em que deve ser executado.

Para a geração de símbolos intermediários e labels, utilize dois números globais incrementais, e imprima um nome especial seguido desse número em um buffer, para gerar os nomes dos novos símbolos, como por exemplo “__temp0”, “__temp1”, etc… Faça rotinas makeTemp e makeLabel para isso.

Para a geração de código, além das rotinas utilitárias de TACs e da rotina recursiva principal que percorre a AST, utilize outras rotinas auxiliares. Isto tem dois motivos: primeiro, a semelhança na geração de código em vários nodos da árvore, especialmente nos operadores aritméticos, relacionais, etc.... Em segundo lugar, use rotinas auxiliares para evitar que a rotina recursiva fique grande. Ela deve fazer “switch(node-->type)” e chamar uma rotina auxiliar de geração de código para cada tipo (ou grupo) de instrução.

Além das TACs que irão gerar instruções, você deve usar dois tipos de TACs utilitárias, uma TAC_SYMBOL, que somente facilita a recursão, mas não irá gerar nenhum código assembly, e TAC_LABEL, que marca um rótulo mas funciona como instrução separada, facilitando o isolamento das tarefas de geração do código. Ou seja, você não precisa saber para qual instrução deve saltar, mas sim apenas para qual ponto, e a instrução será a próxima encadeada.

Lista de TACs

Para auxiliar na sua tarefa segue aqui uma lista sugerida de TACs, a qual pode ser completada.

TAC_SYMBOL, TAC_MOVE, TAC_ADD, TAC_MUL, ...,

TAC_LABEL, TAC_BEGINFUN, TAC_ENDFUN, TAC_IFZ, TAC_JUMP, TAC_CALL, TAC_ARG, TAC_RET, TAC_PRINT, TAC_READ …

Controle e organização do seu código fonte

Você deve seguir as mesmas regras das etapas anteriores para organizar o código, permitir compilação com make, permitir que o código seja executado, e esteja disponível.

Delimitações da linguagem para facilitar a implementação do trabalho.

Uma vez que estamos abrindo mão da etapa Análise Semântica, para este trabalho, você pode, se desejar, considerar as seguintes restrições do programa a ser apresentado ao nosso compilador:

1) Variáveis serão tidas sempre como globais. Com isso, não há possibilidades de um mesmo nome de identificador estar associado a mais de um tipo.
2) Funções e procedimentos criados pelo programador da linguagem não vão aceitar parâmetros. De modo que não será necessário preocupar-se em validar quantidade e tipos dos parâmetros. Entretanto, para os comandos e funções da linguagem (as que já são conhecidas) essa restrição de não haver parâmetros, não se aplica.
3) Considerar o comando while como sendo o disponível para o programador, quando ele desejar usar repetições.
4) Considerar o comando if (e if else) como sendo o disponível para o programador, quando ele desejar usar condicionais.

Equipes

O trabalho pode ser feito em grupos com no máximo 2 alunos.

O que deve ser entregue:

Além da entrega do código fonte, na plataforma do colabweb, o aluno deverá produzir:

1. **Manual do usuário** (uma página) Num arquivo chamado **mu.txt** ou **mu.doc**, contendo uma explicação de como se utilizar o analisador (explicar o formato da entrada e da saída do programa).
2. **um vídeo (ou mais)** explicando como as funções foram pensadas. As principais funções devem também ser explicadas. No vídeo, há a necessidade de apresentar o código fonte, a compilação e a execução do programa. Não há necessidade do aluno aparecer no vídeo.

Os vídeos devem ser postados em uma plataforma de vídeo (youtube, por exemplo), de modo que o professor possa acessar, e fazer parte da avaliação do trabalho. Para gravar pode usar serviços gratuitos e online. Por exemplo:

https://online-screen-recorder.com/pt

https://www.veed.io/pt-BR

https://streamyard.com/

ou se quiser, há aplicativos que podem ser baixados:

https://www.movavi.com/pt/learning-portal/gravadores-de-video.html

Atenção: Os vídeos não precisam ter alta produção. Para subir os vídeos para o Youtube, há uma necessidade de ajuste no seu perfil do youtube, com pelo menos 24h de antecedência. Portanto, sugiro que esse ajuste seja feito brevemente.

A nota será composta assim

- De 0 a 0,5 pontos pelo estilo de programação (nomes bem definidos, lugares de declarações, comentários).
- De 0 a 0,5 ponto pela compilação.
- De 0 a 0,5 ponto pelo formato de apresentação dos resultados
- De 0 a 7 pontos pela solução apresentada.
- De 0 a 1,5 pontos pelas informações dos vídeos.

Comentários Gerais

1. Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.
2. Clareza, indentação e comentários no programa também vão valer pontos.
3. Trabalhos copiados serão penalizados conforme anunciado