

Especificação da Etapa 6 do Projeto de Compilador

Suporte à Execução

A sexta etapa do trabalho de implementação de um compilador para a **Linguagem IKS** consiste na geração de código intermediário para o suporte à execução. A parte principal consiste no código para gerenciar o fluxo de controle através da pilha de ativação e os seus registros de ativação. Utilizaremos como representação intermediária a **Linguagem ILOC**, descrita em detalhes no apêndice A de *Engineering a Compiler* [1], mas com o essencial descrito na definição da quinta etapa do projeto de compiladores. Os registros de ativação deverão conter pelo menos as seguintes informações (em ordem arbitrária):

- Temporários (caso sejam necessários)
- Variáveis locais
- Estado da Máquina
- Vínculo estático
- Vínculo dinâmico
- Valor retornado
- Parâmetros formais (argumentos)
- Endereço de retorno

1 Funcionalidades Necessárias

Os resultados compreendem a *correção dos problemas encontrados na etapa anterior* e as funcionalidades seguintes:

1.1 Registros de Ativação

Deve ser definida uma ordem dos dados dentro de um registro de ativação. Esta ordem deve ser obedecida no momento de criar o registro de ativação de cada função da Linguagem **IKS**. Cada função deve ter o seu padrão de registro de ativação, de acordo com seus parâmetros formais e variáveis locais.

1.2 Gerenciamento da Pilha

O código a ser embutido no programa para o suporte ao ambiente de execução deve ser capaz de gerenciar o fluxo de controle do programa através da pilha de ativação. Assuma que existam registradores específicos para tal: como o *stack pointer* (sp) e o *frame pointer* (fp). Os endereços de vínculo estático e dinâmico de cada registro de ativação devem ser corretamente calculados.

1.3 Sequência de Ativação

O programa deve ter código específico para a chamada e o retorno de cada subprograma da Linguagem **IKS**. O código de chamada deve ser capaz de criar corretamente, na pilha, uma instância do registro de ativação de um subprograma apontando o fluxo de execução para o código da função chamada no segmento de código. De maneira equivalente, o código de retorno deve ser capaz de atualizar a pilha e retornar o fluxo de execução para a função chamadora, no ponto imediatamente após a instrução de chamada do subprograma. Cada subprograma deve ter um rótulo específico indicando a sua primeira instrução.

1.4 Passagem de Parâmetros

A linguagem **IKS** possui apenas passagem de parâmetro por valor com semântica de entrada, que pode ser implementado realizando uma cópia do parâmetro real para o parâmetro formal. Lembre-se que a Linguagem **IKS** não aceita passagem de arranjos como parâmetros, somente variáveis de tipo simples.

Entrada e Saída Padrão

Organize a sua solução para que o compilador leia o programa em **IKS** da entrada padrão e gere o programa em ILOC na saída padrão. Dessa forma, pode-se realizar o seguinte comando (considerando que **main** é o binário do compilador):

```
./main < entrada.iks > saida.iloc
```

Onde **entrada.iks** contém um programa em **IKS**, e **saida.iloc** contém o programa em ILOC correspondente.

2 Controle e Organização da Solução

A função **main** deve estar em um arquivo chamado **main.c**. Outros arquivos fontes são encorajados de forma a manter a modularidade do código fonte. A entrada para o *bison* deve estar em um arquivo com o nome **parser.y**. A entrada para o *flex* deve estar em um arquivo com o nome **scanner.l**.

2.1 Git e Cmake

A solução desta etapa do projeto de compiladores deve ser feita sobre a etapa anterior. Cada ação de commit deve vir com mensagens significativas explicando a mudança feita. Todos os membros do grupo devem ter feito ações de commit, pelo fato deste trabalho ser colaborativo. Estas duas ações – mensagens de commit e quem fez o commit – serão obtidas pelo professor através do comando **git log** na raiz do repositório solução do grupo.

A solução do grupo deve partir do código da etapa anterior do projeto de compilador do mesmo grupo. Novos arquivos de código fonte podem ser adicionados, modificando o arquivo **CMakeLists.txt**, para que ele seja incluído no processo de compilação do analisador sintático.

2.2 Documentação do Código e Testes

Todas as funções devem estar documentadas. A escolha do sistema de documentação fica a critério do grupo e será igualmente avaliada. Uma opção é utilizar **doxygen**. A solução deve vir acompanhada com testes exaustivos que verificam o bom funcionamento da solução desta etapa.

3 Atualizações e Dicas

Verifique regularmente o Moodle da disciplina e o final deste documento para informar-se de alguma eventual atualização que se faça necessária ou dicas sobre estratégias que o ajudem a resolver problemas particulares. Em caso de dúvida, não hesite em consultar o professor.

Referências

[1] Keith D. Cooper and Linda Torczon. *Engineering a Compiler*. Morgan Kaufmann, 2nd edition, 2012.