

## Especificação da Etapa 4 do Projeto de Compilador

### Análise Semântica

A quarta etapa do trabalho de implementação de um compilador para a **Linguagem IKS** consiste nas primeiras verificações semânticas durante o processo de compilação. Estas verificações farão parte do sistema de tipos da Linguagem **IKS**, que tem como característica a existência de coerção entre os tipos inteiros, flutuantes e booleanos de acordo com um conjunto de regras detalhado na Seção 2 deste documento. Toda a verificação de tipos é feita de forma estática, em tempo de compilação, e deve considerar os escopos local e global. Finalmente, todos os nós da Árvore Sintática Abstrata (AST), gerada na etapa anterior, devem ter obrigatoriamente um campo que indica o seu tipo. O tipo de um determinado nó da AST pode, em algumas situações, não ser definido diretamente. Nestes casos, seu tipo deve ser inferido de acordo com as regras de inferência da linguagem (apresentadas na Seção 2). Uma série de testes de coerência comportamental das construções sintáticas reconhecidas e representadas na AST devem ser feitas nesta etapa. A Seção 1, a seguir, apresenta as funcionalidades necessárias que devem ser apresentadas nesta etapa. A Seção 3 apresenta os códigos de retorno para os possíveis erros semânticos encontrados.

## 1 Funcionalidades necessárias

Os resultados compreendem *a correção dos problemas encontrados na etapa anterior* e estas funcionalidades.

### 1.1 Verificação de declarações

Todos os identificadores devem ter sido declarados no momento do seu uso, seja como variável, como vetor ou como função. Essa verificação de declaração prévia deve considerar os dois escopos da linguagem (local e global). Todas as entradas na tabela de símbolos devem ter um tipo associado conforme a declaração, verificando-se se não houve dupla declaração ou se o símbolo não foi declarado.

### 1.2 Uso correto de identificadores

O uso de identificadores deve ser compatível com sua declaração e com seu tipo. Variáveis somente podem ser usadas sem indexação, vetores somente podem ser utilizados com indexação, e funções apenas devem ser usadas como chamada de função, isto é, seguidas da lista de argumentos entre parênteses.

### 1.3 Tipos e tamanho dos dados

Uma declaração de variável deve permitir ao compilador definir o tipo e o tamanho (descrito na Seção 2) da variável na sua entrada na tabela de símbolos. Com o auxílio dessa informação, quando necessário, os tipos de dados corretos devem ser inferidos onde forem usados, em expressões aritméticas, relacionais, lógicas, ou para índices de vetores. Isso implica que todos os nós da AST são candidatos a terem um tipo definido de acordo com as regras de inferência de tipos. Esse processo de inferência está descrito na Seção 2.

### 1.4 Coerção de tipos

Os tipos inteiro, flutuante e booleanos podem sofrer coerção de acordo com o conjunto de regras apresentados na Seção 2 deste documento. A solução desta etapa deve marcar todos os nós da AST onde uma coerção deverá acontecer no momento da geração de código. Note que a coerção em si ainda não deve acontecer, apenas deve-se detectar e anotar na AST qual coerção deverá acontecer.

## 1.5 Argumentos e parâmetros

A lista de argumentos fornecidos em uma chamada de função deve ser verificada contra a lista de parâmetros formais na declaração da mesma função. Cada chamada de função deve prover um argumento para cada parâmetro, e ter o seu tipo compatível. A compatibilidade de tipos da linguagem é apresentada na Seção 2 deste documento.

## 1.6 Verificação de tipos em comandos

Todos os comandos simples da linguagem deve ser verificados semanticamente. O comando **input** somente aceita identificadores de qualquer tipo como parâmetro; o comando **output** aceita um literal **string** ou uma expressão aritmética a ser impressa. O comando de retorno **return** deve ser seguido obrigatoriamente por uma expressão cujo tipo é compatível com o tipo de retorno da função. Prevalece o tipo do identificador em um comando de atribuição.

# 2 Sistema de tipos da Linguagem IKS

## 2.1 Coerção

As regras de coerção de tipos da Linguagem **IKS** são as seguintes:

- Não há coerção para os tipos **string** e **char**
- Um tipo **int** pode ser convertido implicitamente para **float** e para **bool**
- Um tipo **bool** pode ser convertido implicitamente para **float** e para **int**
- Um tipo **float** pode ser convertido implicitamente para **int** e para **bool**, perdendo precisão

## 2.2 Inferência

As regras de inferência de tipos da Linguagem **IKS** são as seguintes:

- A partir de **int** e **int**, infere-se **int**
- A partir de **float** e **float**, infere-se **float**
- A partir de **bool** e **bool**, infere-se **bool**
- A partir de **float** e **int**, infere-se **float**
- A partir de **bool** e **int**, infere-se **int**
- A partir de **bool** e **float**, infere-se **float**

## 2.3 Tamanho

O tamanho dos tipos da linguagem **IKS** é definido da seguinte forma:

- Um **char** ocupa 1 byte
- Um **string** ocupa 1 byte para cada caractere
- Um **int** ocupa 4 bytes

- Um float ocupa 8 bytes
- Um bool ocupa 1 byte
- Um vetor ocupa o seu tamanho vezes o seu tipo

## 2.4 Código de tipos

Para simplificar a codificação do compilador, sugere-se a utilização das seguintes definições:

```
#define IKS_INT      1
#define IKS_FLOAT    2
#define IKS_CHAR     3
#define IKS_STRING   4
#define IKS_BOOL     5
```

## 3 Códigos de retorno para erros semânticos

A lista abaixo apresenta os códigos de retorno que devem ser utilizados quando o compilador encontrar erros semânticos. O programa deve lançar uma chamada exit utilizando esses códigos imediatamente após a impressão da linha que descreve o erro encontrado. Outros erros podem ser criados pelo grupo, bastante para tal adicioná-los ao final desta lista, informando o professor da existência deles no momento da submissão.

```
#define IKS_SUCCESS      0 //caso não houver nenhum tipo de erro

/* Verificação de declarações */
#define IKS_ERROR_UNDECLARED 1 //identificador não declarado
#define IKS_ERROR_DECLARED   2 //identificador já declarado

/* Uso correto de identificadores */
#define IKS_ERROR_VARIABLE   3 //identificador deve ser utilizado como variável
#define IKS_ERROR_VECTOR     4 //identificador deve ser utilizado como vetor
#define IKS_ERROR_FUNCTION   5 //identificador deve ser utilizado como função

/* Tipos e tamanho de dados */
#define IKS_ERROR_WRONG_TYPE  6 //tipos incompatíveis
#define IKS_ERROR_STRING_TO_X 7 //coerção impossível do tipo string
#define IKS_ERROR_CHAR_TO_X   8 //coerção impossível do tipo char

/* Argumentos e parâmetros */
#define IKS_ERROR_MISSING_ARGS 9 //faltam argumentos
#define IKS_ERROR_EXCESS_ARGS  10 //sobram argumentos
#define IKS_ERROR_WRONG_TYPE_ARGS 11 //argumentos incompatíveis

/* Verificação de tipos em comandos */
#define IKS_ERROR_WRONG_PAR_INPUT 12 //parâmetro não é identificador
#define IKS_ERROR_WRONG_PAR_OUTPUT 13 //parâmetro não é literal string ou expressão
#define IKS_ERROR_WRONG_PAR_RETURN 14 //parâmetro não é expressão compatível com tipo do retorno
```

## 4 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**.

## 4.1 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.

## 4.2 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.

# 5 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.

# 6 Log de mudanças deste documento

**25/09/2013 às 07h14.** Códigos de retorno para erros semânticos (Seção 3)

**23/09/2013 às 13h25.** Prevalece o tipo do identificador em um comando de atribuição (Seção 1.6)

**23/09/2013 às 13h20.** Alteração das definições do preprocessador para os tipos da linguagem (Seção 2.4)

**22/09/2013 às 00h00.** Remoção do escopo aninhado (Seção 1)