Compilador LAMP

Analisador léxico e Tabela de simbolos

O compilador consiste em um unico arquivo que compila todos os valores necessarios para sua compilação. O nosso compiladore é baseado funções estaticas que são chamadas pela main. Possui apenas um cabeçalho `hashtable.h` que é a interface que da forma a nossa tabela de simbolos.

Arquitetura do compilador

O compilador contem funções que são chamadas estaticamentes.

A primeira função é a função `error` que é responsavél por printar a mensagem de erro e parar a execução da compilação.

A função `readin` tem como função abrir o arquivo e popular o nosse raw com aquela string que está no arquivo. Essa função pode lançar diversas exceções e finalizar a compilação.

A função `ident` tem como obrigação buscar os valores dos tokens literais, ou seja, aquelas strings que não são simbolos logicos, como `&&`, etc. Essa função tem como finalidade mostrar tanto os tokens reservados, como as variáveis - denominadas identificadores.

A função `number` tem como responsabilidade a construção dos valores numéricos inteiros ou flutuantes.

A função `comment` tem como responsabilidade a leitura de um comentário.

A função `literal` tem como responsabilidade a leitura de uma string constante.

A função `parser` é o parser do programa que usa o lexico como função - aqui ele não tem um função bem definida, ele apenas printa os tokens lidos.

A função `words` consiste nas palavras reservadas, que usam a tabela de simbolos para bloquear essas palavras.

Execução

```
O programa faz uso de Makefile, logo para uma pre compilação usa-se
```

`make all` para a compilação e para a execução dos testes usa-se `make test`. Como nessa primeira entrega o make test vai printar toda a tabela de simbolos com todos os token pre definidos e os tokens escaneados.

Execução de um arquivo

```
Caso seja nescessário a executção de apenas um arquivo melo compilador,
```

deverá executar o comand `make ARGS=file`; - O arquivo retornará a string Compilation Succesfull. Exiting. se o arquivo compilar corretamente sem nenhum erro de sintaxe. - Caso retorne algum erro retornará qual token e e onde aconteceu o erro.

Mudanças de codigo

Para uma melhor legibilidade ocorreram alterações na forma como o compilador interpreta os erros, e com isso melhorou a leitura dos mesmos.

Encontramos um erro de interpretação na forma como o comentário era lido e não retornava erro quando o mesmo nao se fechava. Com isso resultou numa modificação de codigo.

Fora adicionado mais 4 funções para a anasile semantica do código.

2 para extrações diretas da variáveis e seus tipos, e outras 2 para a comparação e uma para melhorar o controle dos condicionais.

Mudanças na gramatica

Não fora necessários nenhuma mudanças na gramática, porém os casos de testes foram modificados.

Teste de compilação

A execução de todos os testes serão executados e verificados se existe algum erro.

> Todos os testes serão executados com base nos arquivos da entrega 2.

Teste 1

```
/* Test */
program teste1
begin
    a is int;
    b is float;
    write(a+b);
end.
```

> O teste 1 não contém nenhum erro semantico.

Teste 2

```
program teste1
begin
 a, b is int;
 result is int;
 a,x is float;
 a = 12*a;
 x = 12.0;
 read (a);
 read (b);
 read (c);
 result = (a*b + 1) / (c+2);
 write ({Resultado: });
 write (result);
end.
```

> Contém um erro de varável não declarada na primeira execução.

```
./main tests/test2.lamp
lamp exception near line: 11
Exception: variable not declared

Makefile:20: recipe for target 'run' failed
make: [run] Error 1 (ignored)
```

A correção consiste em adicionar a variável diretamente em uma declaração e com isso já temos o resultado que gostariámos.

Teste 3

```
program teste2
begin
 a, b, c is int;
 d, var is float;
 teste2 = 1;
 read (a);
 b = a * a;
 c = b + a/2 * (35/b);
 write (c);
 val = 34.2;
 c = val + 2.2 + a;
 write (val);
end.
```

> A variável val não foi declarada, com isso não é possivel terminar a execução.

```
 program teste2
begin
 a, b, c is int;
 d, var, val is float;
 teste2 = 1;
 read (a);
 b = a * a;
 c = b + a/2 * (35/b);
 write (c);
 val = 34.2;
 c = val + 2.2 + a;
 write (val);
end.
```

Adicionar a declaração de vão, já consiste inteiramente na resolução do problema.

Teste 4

```
 program test4
 begin
 a, aux is int;
 b is float;
 b = 0;
 read(a);
 read(b);
 if (a>b) then
 aux = b;
 b = a;
 a = aux;
 end
 write(a);
 write(b);
 end.
```

Não contem nenhum erro semantico.

Teste 5

```
 program teste4
/* Teste4 do meu compilador */
begin
 pontuacao, pontuacaoMaxina, disponibilidade is int;
 pontuacaoMinima is char;
 pontuacaoMinima = 50;
 pontuacaoMaxima = 100;
 write({Pontuacao do candidato: });
 read(pontuacao);
 write({Disponibilidade do candidato: });
 read(disponibilidade);

 while (pontuacao>0) && (pontuacao<=pontuacaoMaxima) do
 if ((pontuacao > pontuacaoMinima) && (disponibilidade==1)) then
 write({Candidato aprovado.});
 else
 write({Candidato reprovado.})
 end
 write({Pontuacao do candidato: });
 read(pontuacao);
 write({Disponibilidade do candidato: });
 read(disponibilidade);
 end
end.
```

> A declaração erronea da variável `pontuacaoMaxina` faz a execução semantica não funcionar.

```
./main tests/test5.lamp
lamp exception near line: 13
Exception: variable not declared

Makefile:20: recipe for target 'run' failed
make: [run] Error 1 (ignored)
```

A simples correção do nome da variável já faz ele compilar corretamente.

```
program teste4
/* Teste4 do meu compilador */
begin
 pontuacao, pontuacaoMaxima, disponibilidade is int;
 pontuacaoMinima is char;
 pontuacaoMinima = 50;
 pontuacaoMaxima = 100;
 write({Pontuacao do candidato: });
 read(pontuacao);
 write({Disponibilidade do candidato: });
 read(disponibilidade);

 while (pontuacao>0) && (pontuacao<=pontuacaoMaxima) do
 if ((pontuacao > pontuacaoMinima) && (disponibilidade==1)) then
 write({Candidato aprovado.});
 else
 write({Candidato reprovado.})
 end
 write({Pontuacao do candidato: });
 read(pontuacao);
 write({Disponibilidade do candidato: });
 read(disponibilidade);
 end
end.
```

Teste 6

Não contém erros, com isso compila corretamente.

```
 /* Teste do meu compilador */
program teste5
begin
 a, b, c, maior is int;
 outro is char;
 repeat
 write({A});
 read(a);
 write({B});
 read(b);
 write({C});
 read(c);
 if ( (a>b) && (a>c) ) then
 maior = a;
 end
 if (b>c) then
 maior = b;
 else
 maior = c;
 end
 write({Maior valor:});
 write (maior);
 write ({Outro? (S/N)});
 read(outro);
 until (outro == 'N' || outro == 'n')
end.
```

Teste 7

Compila corretamente.

```
 /* Test */
program teste1
begin
    a is int;
    b is float;
    write(a+b);
end.
```

Considerações

O codigo está em um arquivo apenas, dado que queriamos experimentar uma implementação em C pois precisariamos da mesma para algumas materias no futuro do nosso curso e não tivemos muito contato com a mesma. Com isso o codigo pode conter duplicações e problemas de padrões de C dado o contexto que cada um de nós (Lara e João) estamos inseridos.

Autores João Victor Mazagão e Lara Loures