Trabalho: Implementação de um Compilador para a Linguagem C

Parte 1: Construção do Analisador Léxico

Gerar as expressões regulares para cada token referente a linguagem C conforme descrita abaixo e de acordo com o código de exemplo.

1. Palavras Reservadas e Símbolos na Linguagem C

Palavras Reservadas:

auto, break, case, char, const, continue, default, do, double, else, enum, extern,

float, for, goto, if, int, long, register, return, short, signed, sizeof, static,

struct, switch, typedef, union, unsigned, void, volatile, while

Explicação das Categorias:

- **Tipos de dados primitivos:** `char`, `double`, `float`, `int`, `long`, `short`, `signed`, `unsigned`, `void`
- **Controle de fluxo:** `if`, `else`, `switch`, `case`, `default`, `while`, `do`, `for`, `break`, `continue`, `goto`
- **Modificadores de armazenamento:** `auto`, `extern`, `register`, `static`, `typedef`
- **Operadores e manipulação de memória:** `sizeof`
- **Manipulação de estrutura de dados:** `struct`, `union`, `enum`
- **Retorno de função:** `return`
- **Modificadores de variável:** `const`, `volatile`

Operadores e Símbolos:

Aritméticos: +, -, *, /, %, ++, --
Relacionais: ==, !=, >, <, >=, <=
Lógicos: &&, ||, !
Bitwise: &, |, ^, ~, <<, >>
Atribuição: =, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
Outros: sizeof, ?:, ., ->, [ ], { }, ( ), ,

#include <stdio.h>

#include <stdlib.h>

#define MAX ←tratar **#define** para definir constantes.

Comentários:

// alguma coisa ou

/* alguma coisa */ pode ser uma ou mais linhas.

Array:

int array[5] = {1, 2, 3, 4, 5}; // array

int array[5]; // array

int array[]; //

int *arr;

arr = (int *)malloc(n * sizeof(int));

free(arr);

Definicao de escopo: { e }

```
{
  int soma4= 10;
   odd_count = soma4 * 10 + 34 + 45 - 78;
}
```

2. Descrição do Trabalho

O objetivo deste trabalho é implementar um compilador para a linguagem C, considerando palavras reservadas, operadores, estruturas de controle e atribuições. Não será tratado o uso de funções.
O trabalho será dividido em três partes:

Parte 1: Definição de Expressões Regulares para Tokens

Na primeira parte, o aluno deverá definir expressões regulares para os tokens da linguagem, incluindo:
- Palavras reservadas
- Identificadores
- Números inteiros e de ponto flutuante
- Strings e caracteres
- Operadores e símbolos especiais

O objetivo é garantir que o lexer consiga reconhecer corretamente os tokens da linguagem C.

Parte 2: Construção da Gramática

O aluno deverá construir a gramática da linguagem para:
- Atribuições, alocaões, desalocações
- Estruturas condicionais (if, else)
- Estruturas de repetição (while, do-while, for)
- Operações de comparação, defines, includes

A implementação deve exibir mensagens na tela sempre que entrar em um caso específico, para depuração e validação.

Parte 3: Tabela de Símbolos, Análise Semântica e Código Intermediário

A terceira parte do trabalho envolve a criação da tabela de símbolos, a geração das árvores sintáticas e a análise semântica baseada na tabela de símbolos. Além disso, será necessário gerar um código intermediário para expressões de atribuição.

Exemplo de Código:

```
#include <stdio.h>

#include <stdlib.h>


// Definição de macro

#define MAX 100


// Definição de um struct
```

```
struct Data {
   int id;
   char name[50];
};

// Definição de um union
union Value {
   int i;
   float f;
};

// Enumeração de tipos
enum Color { RED, GREEN, BLUE };

// Protótipo de função
void printArray(int *arr, int size);

int main() {
   // Declaração e inicialização de variáveis de vários tipos
   int x = 10;
   signed int w = -20;
   unsigned int tt = 50;
   long int rt = 100000;
   short int si = 5;
   float f = 3.14;
   double d = 2.718;
```

```
char ch = 'A';
const int constant_var = 500;
volatile int volatile_var = 200;

// Ponteiro e alocação dinâmica
int *ptr = (int *)malloc(MAX * sizeof(int));
if (ptr == NULL) {
   printf("Erro na alocação de memória\n");
   return 1;
}

// Inicialização de matriz
int matrix[3][3] = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9}
};

// Inicializando struct
struct Data data1 = {1, "John"};

// Inicializando union
union Value val;
val.i = 10;

// Inicializando enum
```

```
enum Color color = RED;

// Uso de if-else
if (x > 0) {
   printf("x é positivo.\n");
} else {
   printf("x é negativo ou zero.\n");
}

// Uso de switch-case
switch (color) {
   case RED:
      printf("A cor é Vermelho.\n");
      break;
   case GREEN:
      printf("A cor é Verde.\n");
      break;
   case BLUE:
      printf("A cor é Azul.\n");
      break;
   default:
      printf("Cor desconhecida.\n");
}

// Uso de for
for (int i = 0; i < 5; i++) {
```

```
    printf("For loop - Iteração %d\n", i);
}

// Uso de while
int count = 0;
while (count < 3) {
    printf("While loop - Contagem: %d\n", count);
    count++;
}

// Uso de do-while
count = 0;
do {
    printf("Do-while loop - Contagem: %d\n", count);
    count++;
} while (count < 3);

// Uso de ponteiro em uma função
for (int i = 0; i < 10; i++) {
    ptr[i] = i * 10;
}
printArray(ptr, 10);

// Registro de variável
register int fast_var = 30;
printf("Variável registrada: %d\n", fast_var);
```

```
    // Exemplo de typedef
    typedef unsigned long ULong;
    ULong bigNum = 123456789;
    printf("ULong: %lu\n", bigNum);

    // Liberação de memória alocada
    free(ptr);
    printf("Memória desalocada com sucesso.\n");

    return 0;
}

// Função para imprimir um array usando ponteiro
void printArray(int *arr, int size) {
    printf("Array dinâmico: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

Explicação do Código:

1. **Uso de todas as palavras reservadas do ANSI C**:
   - `auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while`
2. **Utilização de diferentes tipos de dados**:
   - `int`, `signed int`, `unsigned int`, `long int`, `short int`, `float`, `double`, `char`, `const`, `volatile`, `typedef`, `enum`, `union`, `struct`
3. **Ponteiros e alocação dinâmica**:
   - `malloc` para alocar um array dinamicamente.
   - `free` para liberar a memória alocada.
4. **Matrizes e inicialização de valores**:
   - Matriz `matrix[3][3]` com valores pré-definidos.
5. **Uso de controle de fluxo (if, else, switch, while, do while, for)**:
   - `if-else` para verificação de condição.
   - `switch-case` para enumeração de valores.
   - `while`, `do-while` e `for` para iteração.
6. **Função com passagem de ponteiro**:
   - `printArray` imprime os valores armazenados no array dinâmico.
7. **Macro `#define MAX 100`**:
   - Define um valor máximo para alocação dinâmica.
8. **Uso de `register` para otimizar uma variável**:
   - `register int fast_var` declara uma variável que pode ser armazenada em um registrador da CPU.
9. **Uso de `typedef` para simplificar tipos de dados**:
   - `typedef unsigned long ULong;`

Saída Esperada (Exemplo):

x é positivo.

A cor é Vermelho.

For loop - Iteração 0

For loop - Iteração 1

For loop - Iteração 2

For loop - Iteração 3

For loop - Iteração 4

While loop - Contagem: 0

While loop - Contagem: 1

While loop - Contagem: 2

Do-while loop - Contagem: 0

Do-while loop - Contagem: 1

Do-while loop - Contagem: 2

Array dinâmico: 0 10 20 30 40 50 60 70 80 90

Variável registrada: 30

ULong: 123456789

Memória desalocada com sucesso.