Cap 228 - Linguagens Formais, Autômatos e Construção de Compiladores
Computação Aplicada, INPE, 2004 - Prof. Senne

Implementar um compilador para a linguagem **SubC** abaixo, contendo:

- analisador léxico
- analisador sintático recursivo descendente
- geração orientada pela sintaxe de código intermediário
- intérprete para o código intermediário

Gramática da Linguagem **SubC**

<programa> ::= **main()** <bloco>
<bloco> ::= **{** [{ <declaração de variáveis> }] <lista de comandos> **}**
<declaração de variáveis> ::= <tipo> <lista de variáveis> **;**
<tipo> ::= **int** | **float**
<lista de variáveis> ::= <variável> { **,** <variável> }
<variável> ::= <identificador> [**[** <inteiro sem sinal> **]**]
<identificador> ::= <letra> { <letra> | **_** | <dígito> }
<lista de comandos> ::= <comando> { <comando> }
<comando> ::=
    <identificador de variável> **=** <expressão> **;** |
    if (<expressão> **)** <bloco> [**else** <bloco>] |
    while (<expressão> **)** <bloco> |
    printf (<string> [**,** <lista de expressões>] **) ;** |
    scanf (<string> **,** <lista de endereços> **) ;**
<identificador de variável> ::= <identificador> [**[** <expressão inteira> **]**]
<expressão> ::=
    <expressão simples> { <operador relacional> <expressão simples> }
<expressão simples> ::= [<sinal>] <termo> { <operador aditivo> <termo> }
<sinal> ::= **+** | **-**
<termo> ::= <fator> { <operador multiplicativo> <fator> }
<fator> ::=
    <constante sem sinal> |
    <identificador de variável> |
    ! <fator> |
    (<expressão> **)**
<lista de expressões> ::= <expressão> { **,** <expressão> }
<operador aditivo> := **+** | **-** | **||**
<operador multiplicativo> ::= ***** | **/** | **%** | **&&**
<operador relacional> := **==** | **!=** | **<** | **>** | **<=** | **>=**
<string> ::= **“** [<caractere> { <caractere> }] **“**
<lista de endereços> ::= <endereço> { **,** <endereço> }
<endereço> ::= **&**<identificador de variável>

Observações:
- Comentários devem ser delimitados por **/*** ... ***/** ou iniciados por **//** (até o final da linha).
- Os primeiros 30 caracteres dos identificadores são significativos.
- Os símbolos que aparecem em azul são símbolos terminais.
- Na notação BNF da gramática acima, colchetes delimitam partes opcionais e chaves delimitam partes que podem aparecer zero ou mais vezes.

Exemplos de Programas SubC

Programa 1: Mostra a conversão de um número decimal D para um número num sistema de numeração de base B (B ≤ 10).

```
main()
{
  int D,B,k,quoc,resto;
  int num[20];

  printf("Entre com um inteiro e uma base: ");
  scanf("%d %d",&D,&B);
  quoc = D;
  k = 0;
  while (quoc >= B)
  {
    resto = quoc % B;
    quoc = quoc / B;
    num[k] = resto;
    k = k + 1;
  }
  num[k] = quoc;
  printf("Representação de %d na base %d: ",D,B);
  while (k >= 0)
  {
    printf("%d",num[k]);
    k = k – 1;
  }
}
```

Programa 2: Resolve um sistema de N (N ≤ 5) equações lineares cuja matriz dos coeficientes é triangular inferior e armazenada no vetor A. O vetor dos termos independentes é B e X é o vetor solução.

```
main()
{
  int i,j,k,N;
  float soma;
  float A[15];
  float B[5],X[5];
  printf("Número de equações: ");
  scanf("%d",&N);
  i = 1;
  j = 0;
  while (i <= N)
  {
    j = j + i;
    i = i + 1;
  }
  printf("Coeficientes da matriz A (por linha)\n");
  i = 0;
  while (i < j)
  {
    scanf("%f",&A[i]);
```

```
    i = i + 1;
  }
  printf("Vetor B: ");
  i = 0;
  while (i < N)
  {
    scanf("%f",&B[i]);
    i = i + 1;
  }
  // Determinação do vetor solução
  X[0] = B[0]/A[0];
  i = 1;
  k = 1;
  while (i < N)
  {
    soma = 0;
    j = 0;
    while (j < i)
    {
      soma = soma + A[k]*X[j];
      j = j + 1;
      k = k + 1;
    }
    X[i] = (B[i]-soma)/A[k];
    i = i + 1;
    k = k + 1;
  }

  printf("Vetor solução:\n");
  i = 0;
  while (i < N)
  {
    printf("%f\n",X[i]);
    i = i + 1;
  }
}
```

Programa 3: Calcula $E = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}$, para valores dados de x (x > 0) e n.

```
main()
{
  float x;
  int n;

  printf("Entre com x e n");
  scanf("%f %d",&x,&n);
  if (x > 0)
  {
    float E,F,p;
    int i;

    E = 0;
    F = 1;
    p = 1;
```

```
    i = 0;
    while (i <= n)
    {
      E = E + p/F;
      p = p * x;
      i = i + 1;
      F = F * i;
    }
    printf("E = %f\n",E);
  }
  else
    printf("x deve ser positivo\n");
}
```