Universidade Federal do Amazonas - UFAM
Instituto de Computação – IComp, Curso de Ciência da Computação
Compiladores
Projeto I: likeForth → C/C++ (versão 1.1)

Objetivo

Neste projeto você deve escrever um tradutor da linguagem **likeForth** para a linguagem **C/C++**. Ou seja, um conversor de códigos fonte escritos em um dialeto do Forth para códigos C/C++.

Descrição

Como Forth, likeForth é uma linguagem baseada em pilha. Um programa em likeForth é uma sequência de instruções que modificam uma pilha. Cada instrução se refere a uma palavra previamente definida ou a definição de uma nova palavra. Por exemplo, no programa abaixo:

```
3 4 +
```

há três palavras, os números 3 e 4 e o sinal +. Os números 3 e 4 são inseridos na pilha. A palavra + indica que os números no topo da pilha (4, 3; 4 no topo) serão desempilhados, somados (+) e o resultado (7) será inserido na pilha. Assim, dada uma pilha vazia, após a execução de "3 4 +", ela passa a ter um elemento, 7. A seguir, um exemplo mais complexo:

```
: quadrado DUP * ;
3 quadrado
```

Na primeira linha é definida a palavra *quadrado*. Definições são iniciadas com um ':'. A palavra definida será substituída por todas as palavras até um ';'. No exemplo, a definição de quadrado duplica o valor no topo da pilha (palavra *reservada* DUP) e então multiplica os dois elementos do topo (palavra *). Ou seja, após a execução de quadrado, o valor que estava no topo da pilha é substituído por seu quadrado. Na segunda linha, o valor 3 é inserido na pilha e a palavra quadrado é aplicada, de forma que o valor 3 é substituído por 9. A linguagem likeForth conta com uma série de palavras pré-definidas:

Palavra	Definição
: *palavra* Seq ; numero	Define *palavra* como sequencia de outras palavras (Seq) Insere número na pilha de trabalho
. . "string" ? CR	Exibe topo da pilha na tela Exibe string na tela Lê um número do teclado e o insere na pilha Salta uma linha (mesmo que imprimir '\n' no C/C++)
+ - * / % SQRT	Soma, subtração, multiplicação, divisão, resto e raíz quadrada
> < = OR AND NOT EMPTY REMPTY	Empilha 0 ou 1 de acordo com operador maior, menor ou igual Empilha 0 ou 1 de acordo com operador relacional *ou*, *e* e *não* Empilha 1 se pilha de trabalho está vazia (0, se não) Empilha 1 se pilha de resultados vazia (0, se não)
DROP SWAP DUP ROT PICK ROLL >R R> R@	Desempilha valor no topo da pilha (3 4 2 1 -> 4 2 1) Troca valores no topo (3 4 2 1 -> 4 3 2 1) Duplica valor no topo (3 4 2 1 -> 3 3 4 2 1) Rotaciona top-3 valores na pilha (3 4 2 1 -> 4 2 3 1) Copia n-esimo valor pro topo (1 PICK com 5 6 7 -> 6 5 6 7) Move n-esimo valor pro topo (1 ROLL com 5 6 7 -> 6 5 7) Move topo da pilha de trabalho pra pilha de resultados Move topo da pilha de resultados pra pilha de trabalho Copia topo da pilha de trabalho pra pilha de resultados
BEGIN Seq Cond UNTIL N DO Seq LOOP Cond IF Seq1 [ELSE Seq2] THEN	Repete Seq até condição Cond ser satisfeita Repete Seq N vezes (N é valor no topo da pilha visto por DO) Executa Seq1 se Cond é verdadeira (senão executa Seq2)
SRAND RAND	Usa semente arbitrária para serie aleatória Empilha valor aleatório entre 0 e o valor no topo da pilha

Note que palavras definidas pelo programdor em likeForth são sempre iniciadas com letras minúsculas (ex: quadrado, duplicar4vezes), enquanto palavras pré-definidas usam apenas maiúsculas (ex: DUP, ROLL). Números podem ser inteiros (ex: 7) ou reais (ex: 3.14) e podem ser negativos (ex: -7 ou -3.14). Comentários são colocados entre parênteses.

Em geral, todos os comandos consomem os elementos da pilha que utilizam (R@ é uma exceção). Por exemplo, se a pilha tem os elementos 5, 4, 3, após a execução de *, os elementos 5 e 4 são consumidos e substituídos por seu produto, 20. Assim, a pilha fica com os elementos 20 e 3. Operados lógicos e relacionais sempre deixam um valor 0 (falso) ou 1 (verdadeiro) no topo da pilha. Por exemplo, se a pilha tem 10, 11, 3, o operador > deixa o valor 1 (verdadeiro) no topo da pilha pois 11 é maior que 10. A pilha ficaria então com 1, 3.

Palavras PICK e ROLL não consideram o valor no topo da pilha (usando como índice) no cálculo do elemento que devem processar. Por exemplo, dada a pilha 10, 11, 12, 13, as palavras 2 ROLL indicam que o elemento 2 da pilha (elemento 12 já que o elemento 0 é 10) deve ser movido pro topo. Logo a pilha fica com 12, 10, 11, 13 (note que, de fato, a pilha vista por ROLL é 2, 10, 11, 12, 13; uma vez que o topo indica o elemento a ser movido. Esse elemento [2] não é considerado no cálculo da posição).

Apesar de laços aninhados serem possíveis em Forth, eles são evitados em likeForth a não ser que sejam feitos por meio de definições. Por exemplo, `10 DO 4 DO DROP LOOP LOOP` deveria executar `DROP` 40 vezes. Essa instrução é ilegal em likeForth(embora seu tradutor não precise checar isso). Para conseguir o mesmo efeito, o programador deve usar uma outra palavra:

```
: loopInterno ( executa DROP 4 vezes ) 4 DO DROP LOOP ;
10 DO loopInterno LOOP
```

A seguir, temos um exemplo de código em likeForth e seu equivalente em C++. Note que os códigos foram organizados de forma a facilitar a comparação.

Código fonte likeForth	C++

`(`
`  Codigo em LikeForth`
`)`
 	
`: inverte (a b c -- c b a) SWAP 2 ROLL ;`
 	
`: delta (a b c -- SQRT(b * b - 4 * a * c))`
`  SWAP DUP *   ( a b c -> b*b a c)`
`  ROT * 4 *    ( -> 4*a*c b*b )`
`  -            ( -> b*b - 4*a*c )`
`  DUP`
`  0 < NOT`
`  IF`
`    SQRT       ( se delta >= 0, SQRT(delta) )`
`  ELSE`
`    DROP -1    ( senao deixa soh -1 na pilha )`
`  THEN`
`;`
 	
`: raiz1 (a b c -- -b/2a ) 2 * SWAP -1 * / ;`
 	
`: raiz2  (a b c -- (-b+-SQRT(b*b-4*a*c))/2a )`
	`#include "iostream"`
	`#include "lforth.h"`
	
	`using namespace std;`
	
	`void inverte(likeforth &lf) {`
	`    lf.swap(); lf.push(2); lf.roll();`
	`}`
	
	`void delta(likeforth &lf) {`
	`    lf.swap(); lf.dup(); lf.mul();`
	`    lf.rot(); lf.mul(); lf.push(4); lf.mul();`
	`    lf.sub();`
	`    lf.dup();`
	`    lf.push(0); lf.clt(); lf.cnot();`
	`    if (lf.testok()) {`
	`        lf.ssqrt();`
	`    } else {`
	`        lf.drop(); lf.push(-1);`
	`    }`
	`}`
	
	`void raiz1(likeforth &lf) {`
	`    lf.push(2); lf.mul(); lf.swap();`
	`    lf.push(-1); lf.mul(); lf.div();`
	`}`
	
	`void raiz2(likeforth &lf) {`

```
  DUP 2 *
  3 ROLL 3 ROLL 3 ROLL
  1 PICK
  -1 *
  3 ROLL 3 ROLL 3 ROLL
  delta
  SWAP
  1 PICK 1 PICK
  SWAP - >R
  SWAP + >R
  DUP R> SWAP / SWAP
  R> SWAP /
;




: raizes ( a b c -- 0,1ou2 raizes na pilha )
  3 DO 2 PICK LOOP
  delta
  DUP 0 =
  IF            ( se delta == 0, 1 raiz )
    DROP raiz1
    SWAP DROP
  ELSE
    0 >
    IF          ( se delta > 0, 2 raizes )
      raiz2
    ELSE        ( senao, pilha vazia )
      BEGIN DROP EMPTY UNTIL
    THEN
  THEN
;









(--
  Le 3 valores do teclado e resolve
    a eq de 2o grau correspondente.
    Marco Cristo, 2014, Compiladores
--)

."valor de a: " ?
."valor de b: " ?
."valor de c: " ?
inverte
raizes

EMPTY
IF
  ."Nao ha solucoes!" CR
ELSE
  ."Raiz 1: " . CR
  EMPTY NOT
  IF
    ."Raiz 2: " . CR
  THEN
THEN
```

```
    lf.dup(); lf.push(2); lf.mul();
    lf.push(3); lf.roll();
    lf.push(3); lf.roll();
    lf.push(3); lf.roll();
    lf.push(1); lf.pick();
    lf.push(-1); lf.mul();
    lf.push(3); lf.roll();
    lf.push(3); lf.roll();
    lf.push(3); lf.roll();
    delta(lf);
    lf.swap();
    lf.push(1); lf.pick();
    lf.push(1); lf.pick();
    lf.swap(); lf.sub(); lf.rpush();
    lf.swap(); lf.add(); lf.rpush();
    lf.dup(); lf.rdrop(); lf.swap();
    lf.div(); lf.swap();
    lf.rdrop(); lf.swap(); lf.div();
}

void raizes(likeforth &lf) {
    lf.push(3);
    int __vtop = lf.top(); lf.drop();
    for(int i = 0; i < __vtop; i ++) {
        lf.push(2);
        lf.pick();
    }
    delta(lf);
    lf.dup(); lf.push(0); lf.ceq();
    if (lf.testok()) {
        lf.drop(); raiz1(lf);
        lf.swap(); lf.drop();
    } else {
        lf.push(0); lf.cgt();
        if (lf.testok()) {
            raiz2(lf);
        } else {
            do {
                lf.drop();
                lf.empty();
            } while(! lf.testok());
        }
    }
}

int main(void)
{
    likeforth lf;

    cout << "valor de a: ";
    {float val; cin >> val; lf.push(val);}
    cout << "valor de b: ";
    {float val; cin >> val; lf.push(val);}
    cout << "valor de c: ";
    {float val; cin >> val; lf.push(val);}
    inverte(lf);
    raizes(lf);
    lf.empty();
    if (lf.testok()) {
        cout << "Nao ha solucoes!";
        cout << endl;
    } else {
        cout << "Raiz 1: ";
        lf.dot();
        cout << endl;
        lf.empty(); lf.cnot();
        if (lf.testok()) {
            cout << "Raiz 2: ";
            lf.dot();
            cout << endl;
        }
    }
}
```

Esta tradução consistiu basicamente no uso de uma classe que implementa todos os serviços do likeForth (classe likeforth). Esta classe está descrita no header **likeforth.h** que *não* é fornecida neste projeto. Você *deve* fazer a sua *própria* versão dela. Note ainda que você não precisa traduzir os comentários pro C/C++ nem garantir que DOs aninhados não ocorram.

Avaliação

O seu tradutor será avaliado diretamente pela execução dos códigos gerados. Ou seja, o código gerado em C/C++ será compilado com o gcc/g++ e testado. A avaliação consiste em verificar se o código em C/C++ faz o que o código original likeForth faria.

Observações

(a) Consulte o professor caso haja dúvidas na especificação informal de likeForth.

(b) Crie casos de teste pro seu tradutor além dos dados em sala de aula, pois casos diferentes dos dados serão usados pelo monitor.

(c) Trabalho feito por, no máximo, dois alunos. Plágio não será tolerado.