INF1018 - Software Básico (2024.1)
Segundo Trabalho

Um compilador para uma Linguagem Básica

O objetivo deste trabalho é desenvolver, em C, uma função `compilaLinB` que implementa um pequeno gerador de código (um "micro-compilador") para uma linguagem de programação básica, chamada LinB.

A função `compilaLinB` deverá ler um arquivo texto contendo o código fonte de uma função escrita em LinB, gerar o código de máquina que corresponde à tradução da função contida no arquivo na área passada no segundo parâmetro e retornar um ponteiro para a função gerada.

A indicação é que a função main que chamar compilaLinB declare como variável local um vetor de unsigned char de tamanho apropriado e o passe como segundo parâmetro.

Instruções Gerais

Leia com atenção o enunciado do trabalho e as instruções para a entrega. Em caso de dúvidas, não invente, Pergunte!

- O trabalho deve ser entregue até a data informada no EaD.
- Trabalhos entregues com atraso perderão **um ponto por dia de atraso ou fração**.
- Trabalhos que não compilem **não serão considerados** (ou seja, receberão grau zero).
- Os trabalhos devem preferencialmente ser feitos em grupos de dois alunos
- Alguns grupos poderão ser chamados para apresentações orais / demonstrações dos trabalhos entregues.

A Linguagem Básica

Funções na linguagem LinB contém atribuições, operações aritméticas, instruções de desvio e de retorno.

- Uma atribuição tem a forma

  ```
  varp '<=' expr
  ```

  onde **`varp`** é uma variável local ou um parâmetro e **`expr`** é uma operação aritmética.

- Uma operação aritmética tem a forma

  ```
  varpc op varpc
  ```

  onde **`varpc`** é uma variável local, um parâmetro ou uma constante inteira e `op` é um dos operadores: + - *

- A instrução de desvio tem a forma

  ```
  'if' varp num
  ```

  onde **`varp`** é uma variável local ou um parâmetro e **`num`** é o número da linha no código

fonte para a qual o controle deve ser desviado caso o valor de `varp` seja **diferente de zero**.

- Finalmente, a instrução de retorno tem a forma

  ```
  'ret'
  ```

  Neste caso, a função deverá retornar, e seu valor de retorno é o valor da variável local `v1`.

Na linguagem LinB, as variáveis locais são da forma `vi`, sendo o índice ***i*** utilizado para identificar a variável (ex. `v1, v2`, etc...). A linguagem permite o uso de no máximo 4 variáveis locais.

Da mesma forma, os parâmetros são denotados por `pi`, e podem ser usados no máximo 2 parâmetros (`p1` e `p2`).

Constantes são escritas na forma `$i`, onde `i` é um valor inteiro, com um sinal opcional. Por exemplo, `$10` representa o valor **10** e `$-10` representa o valor **-10**.

Para sua referência, a sintaxe da linguagem LinB pode ser definida formalmente como abaixo usando a notação BNF (para quem tiver interesse, aqui vai uma referência mais extensa sobre BNF).
Note que as cadeias entre ' ' são símbolos terminais da linguagem: **os caracteres ' não aparecem nos comandos!**

func	::=	cmd '\n' | cmd '\n' func
cmd	::=	att | desvio | ret
att	::=	varp '<=' expr
expr	::=	varpc op varpc
varp	::=	'v' num | 'p' num
varpc	::=	varp | '$' snum
op	::=	'+' | '-' | '*'
ret	::=	'ret'
desvio	::=	'if' varp num
num	::=	digito | digito num
snum	::=	[-] num
digito	::=	0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'| '8' | '9'

Alguns Exemplos

Veja a seguir alguns exemplos de funções LinB.

Note que os comentários **não fazem parte da linguagem!** Eles estão incluidos nos exemplos abaixo apenas para facilitar seu entendimento.

- O exemplo abaixo implementa uma função $f(x) = x + 1$

```
v1 <= p1 + $1          // 1: i = x + 1
ret                   // 2: retorna i
```

- O próximo exemplo implementa uma função *f(x,y) = (x+y) * (x-y)*:

```
v1 <= p1 + p2          // 1: i = x + y
p1 <= p1 - p2          // 2: x = x - y
v1 <= v1 * p1          // 3: i = i * x
ret                   // 4: retorna i
```

- O próximo exemplo é uma função fatorial *fat(n)*:

```
v1 <= $0 + $1          // 1: f = 1
if p1 4               // 2: if (n != 0) desvia para linha 4
ret                   // 3: retorna f
v1 <= v1 * p1          // 4: f = f * n
p1 <= p1 - $1          // 5: n = n - 1
if p1 4               // 6: if (n != 0) desvia para linha 4
ret                   // 7: retorna f
```

- Finalmente, uma função *squad(x)* que calcula a soma dos quadrados de 1 a x:

```
v1 <= $0 + $0          // 1: sum = 0
v2 <= $1 + $0          // 2: i = 1
p1 <= p1 + $1          // 3: x = x + 1
v3 <= p1 - v2          // 4: aux = x - i  (será zero se x == i)
if v3 7               // 5: if (aux != 0) desvia para linha 7
ret                   // 6: retorna sum
v3 <= v2 * v2          // 7: aux = i * i
v1 <= v1 + v3          // 8: sum = sum + aux
v2 <= v2 + $1          // 9: i = i + 1
if v2 4               // 10: if (i != 0) desvia para linha 4 (sempre desviará)
```

Implementação e Execução

O que fazer

Você deve desenvolver em C uma função chamada **`compilaLinB`**, que leia um arquivo de entrada contendo o código fonte de uma função na linguagem LinB, gere o código de máquina correspondente no vetor que é passado como segundo parâmetro, e retorne um ponteiro para a função gerada.

O arquivo de entrada terá no máximo 50 linhas, com um comando LinB por linha.

O protótipo de `compilaLinB` é o seguinte:

```
typedef int (*funcp) ();
funcp compilaLinB (FILE *f, unsigned char codigo[]);
```

O parâmetro **`f`** é o descritor de um arquivo texto, **já aberto para leitura**, de onde deve ser lido o código fonte da função escrita em LinB. Note que a função não deve fechar o arquivo! Esses protótipos estão definidos no arquivo compilalinb.h, disponível aqui.

Implementação

A função **`compilaLinB`** armazenará o código gerado na região de memória passada como segundo parâmetro. O endereço retornado por `compilaLinB` será o endereço do início desta memória.

Para cada instrução *LinB* imagine qual uma tradução possível para *assembly*. Além disso, lembre-se que a tradução de uma função *LinB* deve começar com o prólogo usual (preparação do registro de ativação, incluindo o espaço para variáveis locais) e terminar com a finalização padrão (liberação do registro de ativação antes do retorno da função).

O código gerado deverá seguir as convenções de C/Linux quanto à passagem de parâmetros, valor de retorno e salvamento de registradores (se for o caso). As variáveis locais deverão ser alocadas na pilha de execução.

Para ler e interpretar cada linha da linguagem *LinB*, teste se a linha contém cada um dos formatos possíveis.

Não é necessário fazer tratamento de erros no arquivo de entrada, você pode supor que o código fonte LinB desse arquivo está correto. Vale a pena colocar alguns testes para facilitar a própria depuração do seu código, mas as entradas usadas como testes na correção do trabalho **sempre estarão corretas**.

Veja um esboço de código C para fazer a interpretação de código aqui. Lembre-se que você terá que fazer adaptações pois, dentre outros detalhes, essa interpretação **não será feita na *main***!

O código gerado por `compilaLinB` deverá ser um ***código de máquina x86-64***, e não um código fonte assembly. Ou seja, você deverá descobrir o código de máquina que corresponde às instruções de assembly que implementam a tradução das instruções da linguagem *LinB*. Para isso, você pode usar o programa `objdump` e, se necessário, uma documentação das instruções da Intel.

Por exemplo, para descobrir o código gerado por `movl %eax, %ecx`, você pode criar um arquivo meuteste.s contendo apenas essa instrução, traduzi-lo com o gcc (usando a opção -c) para gerar um arquivo objeto `meuteste.o`, e usar o comando

```
objdump -d meuteste.o
```

para ver o código de máquina gerado.

Estratégia de Implementação

Implemente sua solução passo a passo, **testando separadamente cada passo implementado!**

Por exemplo:

1. Compile um arquivo *assembly* contendo uma função bem simples usando:

   ```
   gcc -c code.s
   ```

   (para apenas compilar e não gerar o executável) e depois veja o código de máquina gerado usando:

   ```
   objdump -d code.o
   ```

   Construa uma versão inicial da função **`compilaLinB`**, que coloque no vetor código esse código, bem conhecido, e retorne o endereço desta área.

   Crie uma função `main` e teste essa versão inicial da função (leia o próximo item para ver como fazê-lo).

2. Implemente e **teste** a tradução de uma função bem simples, como o exemplo a seguir (repare que neste caso precisamos apenas de uma atribuição e uma operação de soma de constantes):

   ```
   v1 <= $0 + $1 // 1: i = 1
   ret           // 2: retorna i
   ```

   Pense em quais informações você precisa extrair para poder traduzir as instruções (quais são os operandos, qual é a operação, onde armazenar o resultado da operação).

   Continue sua implementação, implementando **e testando** uma operação por vez. Experimente usar constantes, parâmetros, variáveis locais, e combinações desses tipos como operandos.

Lembre-se que é necessário reservar espaço (na pilha) para as variáveis locais!

3. Deixe para implementar a instrução de desvio apenas quando **todo o resto** estiver funcionando!

   Pense em que informações você precisa guardar para traduzir essas instruções (note que você precisa saber qual o endereço da instrução correspondente à linha para onde o controle deve ser desviado...)

Testando o gerador de código

Você deve criar um arquivo contendo apenas a função `compilaLinB` (e funções auxiliares) e **outro arquivo** com uma função `main` para testá-la.

Sua função *main* deverá abrir um arquivo texto que contém um "programa fonte" na linguagem *LinB* (i.e, uma função *LinB*) e chamar *compilaLinB*, passando o arquivo aberto como argumento juntamente com um ponteiro para a area onde deverá ser gerado o código de máquina.

Em seguida, sua *main* deverá chamar a função retornada por *compilaLinB*, passando os parâmetros apropriados.

Não esqueça de compilar seu programa com

```
gcc -Wall -Wa,--execstack -o seuprograma seuprograma.c
```

para permitir a execução do código de máquina criado por `compila`!

Uma sugestão para testar a chamada de uma função *LinB* com diferentes argumentos, é sua função *main* receber argumentos passados na linha de comando. Para ter acesso a esses argumentos (representados por *strings*), a sua função `main` deve ser declarada como

```
int main(int argc, char *argv[])
```

sendo **`argc`** o número de argumentos fornecidos na linha de comando e **`argv`** um array de ponteiros para *strings* (os argumentos).

Note que o primeiro argumento para `main (argv[0])` é sempre o nome do seu executável. Os parâmetros que deverão ser passados para a função criada por `compilaLinB` serão os argumentos de 1 em diante. Por exemplo, se o arquivo com o código fonte da linguagem LinB for `teste.linb`, e o executável gerado por você para testar se chamar `compilador`, você deve executar o seu programa como mostrado a seguir:

```
./compilador teste.linb
```

Nesse caso, o nome do arquivo estará em `argv[1]` para você poder abri-lo na sua função `main()` que estará em um arquivo separado que **NÃO** será entregue.

Lembre-se que você não deve entregar o programa completo, somente o arquivo `compilalinb.c` deve ser entregue.

Entrega

Deverão ser entregues **via Moodle** dois arquivos:

1. Um arquivo fonte chamado **`compilalinb.c`** , contendo as função **`compilaLinB`** (e funções auxiliares, se for o caso).

- Esse arquivo **não** deve conter a função `main`.
- Coloque no início do arquivo, como comentário, os nomes dos integrantes do grupo da seguinte forma:

```
/* Nome_do_Aluno1 Matricula Turma */
/* Nome_do_Aluno2 Matricula Turma */
```

2. Um arquivo texto, chamado **`relatorio.txt`**, contendo um pequeno relatório.
   - O relatório deverá explicar o que está funcionando e o que não está funcionando. Não é necessário documentar sua função no relatório. Seu código deverá ser claro o suficiente para que isso não seja necessário.
   - O relatório deverá conter também **alguns** exemplos de funções da linguagem *LinB* que você usou para testar o seu trabalho. Mostre tanto as funções *LinB* traduzidas e executadas com sucesso como as que resultaram em erros (se for o caso).
   - Coloque também no relatório o nome dos integrantes do grupo
   - No relatório, inclua a linha de comando de compilação do seu código e a saída gerada pelo compilador. Lembre-se de usar a opção -Wall quando for compilar.

Indique na área de texto da tarefa do Moodle o nome dos integrantes do grupo. Apenas uma entrega é necessária (usando o *login* de um dos integrantes do grupo) se os dois integrantes pertencerem à mesma turma.