INF1018 - Software Básico (2016.2)
Segundo Trabalho

Mais um Gerador de Código

O objetivo deste trabalho é desenvolver, em C, uma função geracod que implementa um pequeno gerador de código (um "micro-compilador") para uma linguagem de programação bastante simples, chamada SBF.
A função geracod deverá ler um arquivo texto contendo o código fonte de uma ou mais funções escritas em SBF e retornar dois endereços:

- o início da região de memória que contém o código de máquina que corresponde à tradução das funções SBF para código de máquina
- o início do código de máquina da última função lida (essa é a função que será chamada externamente).

Deve ser implementada também uma função liberacod, que libera a memória alocada para armazenar o código criado por geracod.

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 meia-noite (24:00) do dia 28 de novembro**.
Trabalhos entregues com atraso perderão **um ponto por dia de atraso**.
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 SBF

Funções na linguagem SBF contém apenas atribuições, operações aritméticas, chamadas de outras funções e retorno condicional. Todas as funções SBF são delimitadas por uma marca de início (function) e uma marca de fim (end).

- Uma atribuição tem a forma **var '=' expr**, onde **var** é uma variável local e **expr** é uma operação aritmética ou uma chamada de função.
- Uma operação aritmética tem a forma **varpc op varpc**, onde **varpc** é uma variável local, o parâmetro da função ou uma constante inteira e op é um dos operadores: + - *
- A instrução de chamada de função tem a forma **'call' num varpc**, onde **num** é um número que indica a função SBF que será chamada, com argumento **varpc** (uma variável local, o parâmetro da função ou uma constante). A primeira função do arquivo de entrada será a de número 0, a segunda a de número 1, e assim por diante. Uma função só pode chamar a si mesma ou funções que apareçam **antes** dela no arquivo de entrada. **A última função do arquivo de entrada é a que será chamada pelo programa principal**.
- A instrução de retorno condicional tem a forma **'ret?' 'varpc varpc**, seu significado é que, se o primeiro operando tiver valor **igual a zero** a função corrente deverá retornar, e o valor de retorno é o segundo operando. Não haverá retorno se o primeiro operando tiver valor diferente de zero.

Na linguagem SBF, as variáveis locais são da forma **vi**, sendo o índice ***i*** utilizado para identificar a variável (ex. v0, v1, etc...). A linguagem permite o uso de no máximo 5 variáveis locais.
Funções SBF recebem apenas um parâmetro, denotado por **p0**.
Na linguagem SBF as 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**.
A sintaxe da linguagem SBF pode ser definida formalmente como abaixo. Note que as cadeias entre ' ' são símbolos terminais da linguagem: os caracteres ' não aparecem nos comandos SBF.

pgm ::= func | func pgm
func ::= header cmds endf
header ::= 'function\n'
endf ::= 'end\n'
cmds ::= cmd'\n' | cmd '\n' cmds
cmd ::= att | ret
att ::= var '=' expr
expr ::= oper | call
oper ::= varpc op varpc
call ::= 'call' num varpc
ret ::= 'ret?' varpc varpc
var ::= 'v' num
varpc ::= var | 'p0' | '$' snum
op ::= '+' | '-' | '*'
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 SBF.

- Um exemplo **muito simples** é uma função SBF que retorna uma constante:

```
function
ret? $0 $1
end
```

- Este exemplo implementa uma função *f(x) = x + 1*.

```
function
v0 = p0 + $1
ret? $0 v0
end
```

- O próximo exemplo é uma função que calcula o fatorial de seu parâmetro:

```
function
ret? p0 $1
v0 = p0 + $0
v1 = v0 - $1
v1 = call 0 v1
v0 = v0 * v1
ret? $0 v0
end
```

- Finalmente, uma função que calcula a soma dos quadrados de 1 até o seu parâmetro, usando uma função auxiliar para calcular o quadrado de um número:

```
function
v0 = p0 * p0
ret? $0 p0
end
function
ret? p0 $0
v0 = p0 - $1
v1 = call 0 p0
v0 = call 1 v0
v0 = v0 + v1
ret? $0 v0
end
```

Implementação e Execução

O que fazer

Você deve desenvolver em C uma função chamada **geracod**, que leia um arquivo de entrada contendo o código fonte de **uma ou mais** funções na linguagem SBF, gere o código de máquina correspondente, e retorne o endereço da região de memória que contém o código gerado e o endereço do início do código da última função definida no arquivo de entrada.
O arquivo de entrada terá no máximo 50 linhas, com um comando SBF por linha.
O protótipo de geracod é o seguinte:

```
typedef int (*funco) (int x);
void geracod (FILE *f, void **code, funcp *entry);
```

O parâmetro **f** é o descritor de um arquivo texto, **já aberto para leitura**, de onde deve ser lido o código fonte SBF.
O parâmetro **code** é um ponteiro para uma variável (do tipo void *) onde deve ser armazenado o endereço da área que contém o código gerado.
O parâmetro **entry** é um ponteiro para uma variável (do tipo "ponteiro para função que recebe inteiro e retorna inteiro") onde deve ser armazenado o endereço do código da função a ser chamada pelo programa principal.
Você deverá também desenvolver uma função que libere a área de memória alocada por geracod, com o protótipo

```
void liberated (void *p);
```

Implementação

A função **geracod** deve alocar um bloco de memória onde armazenará o código gerado (lembre-se que as instruções de máquina ocupam um número variável de bytes na memória!). Os endereços retornados por geracod serão o endereço do início da memória alocada e o endereço de início do código da última função (dentro dessa área, obviamente).
Para cada instrução *SBF* imagine qual uma tradução possível para *assembly*. Além disso, lembre-se que a tradução de uma função *SBF* 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.

Para ler e interpretar cada linha da linguagem *SBF*, 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 SBF 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 geracod 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 *SBF*. 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

Este trabalho não é trivial. Implemente sua solução passo a passo, **testando separadamente cada passo implementado!**
Por exemplo:

1. Compile um arquivo *assembly* contendo uma função bem simples usando: ***minhamaquina> gcc -c code.c*** (para apenas compilar e não gerar o executável) e depois veja o código de máquina gerado usando: ***minhamaquina> gcc -d code.o***.
2. Construa uma versão inicial da função **geracod**, que aloque uma área de memória, coloque lá esse código, bem conhecido, e retorne o endereço da área alocada como os dois retornos de geracod (os retornos são iguais pois só há uma função no código). Crie uma função main e teste essa versão inicial da função (leia o próximo item para ver como fazê-lo). Teste também a sua função de liberação de memória (chamada pela main!)
3. Implemente e **teste** a tradução de uma função SBF bem simples, como a do primeiro exemplo fornecido. Depois teste uma função que retorne o valor do parâmetro de entrada
4. Comece agora a implementação de atribuições e operações aritméticas. Pense em que informações você precisa extrair para poder traduzir as instruções (quais são os operandos, qual é a operação, onde armazenar o resultado da operação). Implemente **e teste** uma operação por vez. Experimente usar constantes, parâmetros, variáveis locais, e combinações desses tipos como operandos.
   Lembre-se que é necessário alocar espaço (na pilha) para as variáveis locais!
5. Deixe para implementar a instrução call apenas quando **todo o resto** estiver funcionando! Pense em que informações você precisa guardar para traduzir completamente essa instrução (note que você precisa saber qual o endereço da função a ser chamada).

Testando o gerador de código

Você deve criar um arquivo contendo as funções geracod e liberacod 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 *SBF* (i.e, uma função *SBF*) e chamar *geracod*, passando o arquivo aberto como primeiro argumento, e os endereços de duas variáveis que receberão os retornos.
Em seguida, sua *main* deverá chamar a função retornada por *geracod* (o segundo retorno), passando o parâmetro apropriado, e imprimir o valor de retorno dessa função (um valor inteiro).
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 geracod!
Uma sugestão para testar a chamada de uma função *SBF* 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. O parâmetro que deverá ser passado para a função criada por geracod será o argumento 1, convertido para um valor inteiro. Para fazer essa conversão você pode usar a função atoi.

Entrega

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

1. Um arquivo fonte chamado **geracod.c** , contendo as funções **geracod** e **liberacod** (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 *SBF* que você usou para testar o seu trabalho. Mostre tanto as funções *SBF* 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

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.