Trabalho Prático N.01 (Extra – 04 pontos)
Entrega: Domingo, 06 de março de 2022 às 23:59

Neste trabalho, você deve escrever um pequeno programa Cool. O objetivo é para familiarizá-lo com a linguagem Cool e dar-lhe experiência com algumas das ferramentas usadas ao longo do semestre. Esta tarefa **não deverá** ser feita em grupo; você deve entregar seu próprio trabalho individual. Todas as demais etapas do Trabalho Prático que virão podem ser feitas em equipes de até quatro alunos ou individualmente.

Uma máquina com apenas uma única pilha para armazenamento é uma **máquina de pilha**. Considere a seguinte linguagem muito primitiva para programação uma máquina de pilha:

Commando	*Significado*
int	empilha o inteiro *int* na pilha
+	empilha o símbolo '+' na pilha
s	empilha o símbolo 's' na pilha
e	avalia a conteúdo do topo da pilha (ver adiante)
d	exibe o conteúdo da pilha
x	encerra a execução

O comando 'd 'simplesmente imprime o conteúdo da pilha, um elemento por linha, começando com o topo da pilha. O comportamento do comando 'e' depende do conteúdo da pilha quando 'e' for emitido:

- Se '+' estiver no topo da pilha, então o '+' é retirado da pilha, os dois inteiros seguintes são desempilhados e adicionados, e o resultado da soma é empilhado de volta na pilha.
- Se 's' está no topo da pilha, então o 's' é desempilhado e os dois itens seguintes são trocados de lugar na pilha.
- Se um inteiro estiver no topo da pilha ou a pilha estiver vazia, a pilha permanecerá inalterada.

Os exemplos a seguir mostram o efeito do comando 'e' em várias situações; o topo da pilha está à esquerda:

Pilha antes	*Pilha Depois*
$+\ 1\ 2\ 5\ s\ldots$	$3\ 5\ s\ldots$
$s\ 1\ +\ +\ 99\ldots$	$+\ 1\ +\ 99$
$1\ +\ 3\ldots$	$1\ +\ 3\ldots$

Você deve implementar um interpretador para essa linguagem em Cool. Entrada para o seu programa é uma série de comandos, um comando por linha. Seu interpretador deve solicitar comandos com `>`. Seu programa não precisa fazer nenhuma verificação de erros: você pode assumir que todos os comandos são válidos e que o número apropriado e o tipo de argumentos estão na pilha para avaliação. Você também pode assumir que os inteiros de entrada são sem sinal (isto é, são todos positivos). Finalmente, seu interpretador deve sair graciosamente; não chame `abort()` depois de receber um comando `x`.

Você está livre para implementar este programa em qualquer estilo que você escolher. No entanto, em preparação para criar um compilador Cool, recomenda-se que você tente desenvolver uma solução orientada a objetos. Uma abordagem é definir uma classe `StackCommand` com um número de operações genéricas e, em seguida, definir subclasses de `StackCommand`, uma para cada tipo de comando da linguagem. Essas subclasses definem operações específicas para cada comando, por exemplo: como avaliar esse comando, como exibir esse comando, etc. Se desejar, você pode usar as classes definidas em `atoi.cl` no subdiretório `examples` para executar a conversão de string para inteiro. Se você encontrar algum outro código no subdiretório `examples` que considere útil, você pode usá-lo também.

Uma solução (padrão de resposta) foi desenvolvida com aproximadamente 200 linhas de código fonte em Cool. Essas informações são fornecidas a você apenas como uma medida aproximada da quantidade de trabalho envolvida na tarefa – sua solução pode ser substancialmente mais curta ou mais longa.

Exemplo de uso

A seguir, um exemplo de compilação e execução da solução (padrão de resposta) desenvolvida.

```
/var/tmp/cool/bin/coolc stack.cl atoi.cl
/var/tmp/cool/bin/spim -file stack.s
SPIM Version 5.6 of January 18, 1995
Copyright 1990-1994 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README a full copyright notice.
Loaded: /home/ff/cs164/lib/trap.handler
>1
>+
>2
>s
>d
s
2
+
1
>e
>e
>d
3
>x
COOL program successfully executed
```

Uma outra possibilidade é salvar a entrada (lista de comandos) em um arquivo e usá-lo como entrada. Você irá encontrar um exemplo em `stack.test` que é utilizado como entrada padrão para teste. Para utilizá-lo, basta usar `make test`, como pode ser vista a seguir.

```
make test
/var/tmp/cool/bin/coolc stack.cl atoi.cl
stack.test
/var/tmp/cool/bin/spim -file stack.s < stack.test
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /var/tmp/cool/lib/trap.handler
>>>>>>>s
2
+
1
>>>3
>>>>>>>s
s
s
1
+
3
>>>>>>>4
>COOL program successfully executed
```

Obtendo e entregando os arquivos do trabalho

Certifique-se que ter realizado a instalação e configuração do material para realização de trabalhos práticos que foi disponibilizado via **Canvas**. Seu sistema deve conter a hierarquia de diretórios necessários em `/var/tmp/cool`. Certifique-se também de ter instalado o simulador `SPIM` (provavelmente o `qt-spim` também funcionará, mas você pode ter que mudar alguns comandos).

Crie em seu diretório local um subdiretório de trabalho denominado `PA1`, em seguida, faça com que ele seja o diretório corrente, isto é `cd PA1`. A partir dele, digite a seguinte linha de comando:

```
make -f /var/tmp/cool/assignments/PA1/Makefile
```

Este comando cria vários arquivos no diretório que você precisará para realizar a atividade prática. Siga as instruções contidas no arquivo `README`.

Depois de concluído o trabalho, você deverá realizar a entrega dessa tarefa. Para tanto, você deve fazer as seguintes atividades:

1. Assegurar-se que que seu código esteja no arquivo `stack.cl` e de de que ele compila e funciona corretamente :-).
2. Responder as três perguntas existentes no final do arquivo `README`.
3. Compactar e codificar o subdiretório `PA1`, por exemplo da seguinte forma:

   ```
   cd ..
   tar cvzf PA1.tar.gz PA1
   uuencode PA1.tar.gz PA1.tar.gz > PA1.u
   rm PA1.tar.gz
   ```

4. Entregar o arquivo `PA1.u` via **SGA**.