VMS, um Simular Visual para a Máquina de Stack Virtual VM

Adriano Campinho (a79032) / Vasco Leitão (a79220)

Pedro Rangel Henriques e José João

Processamento de Linguagens

Introdução

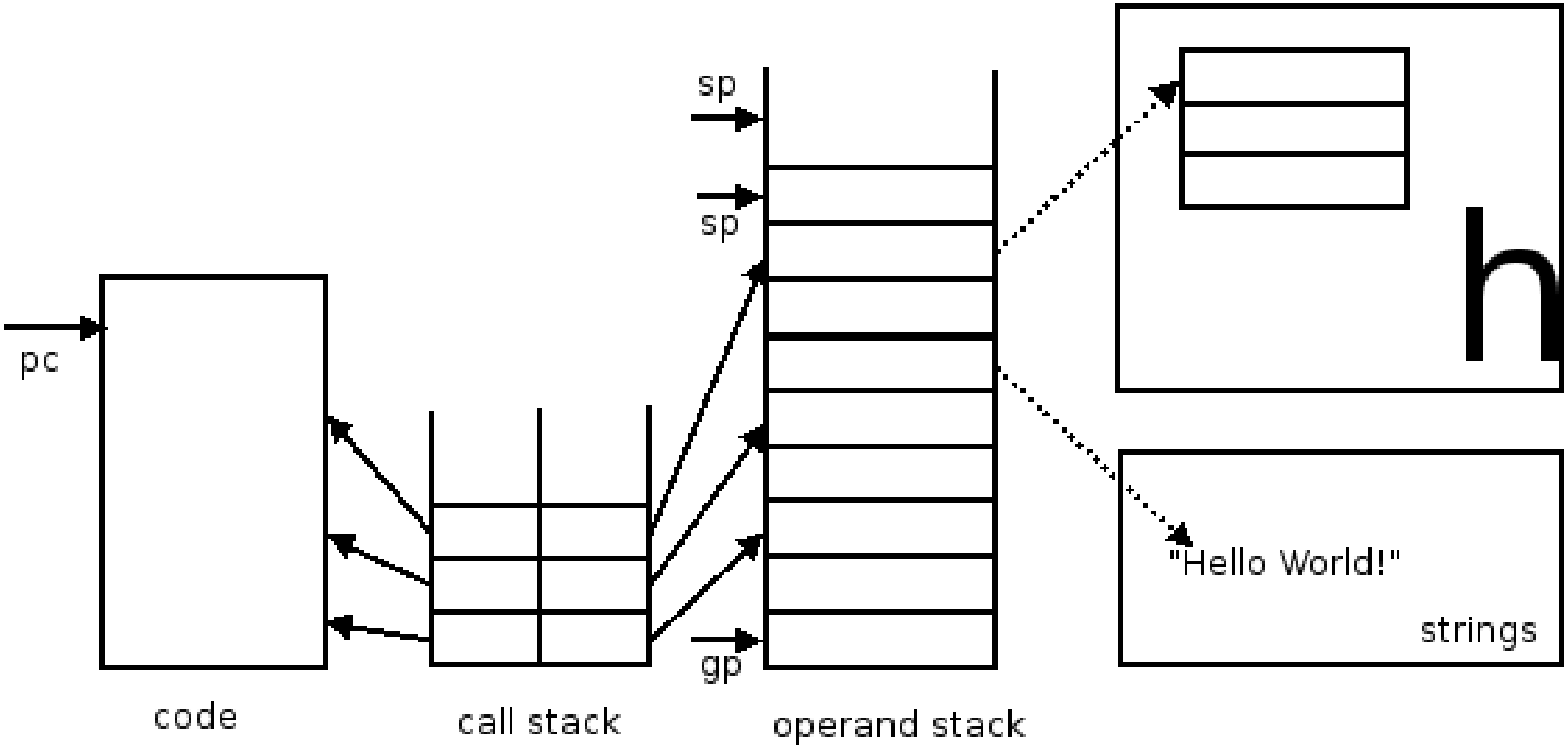
- Nos últimos anos, temos usado nas disciplinas de Processamento de Linguagens (e Compiladores) a máquina virtual VM;
- A VM é programada em Assembly e que disponibiliza um conjunto de instruções máquina mínimo e muito compreensível, o que a torna pedagogicamente muito relevante como máquina objeto (destino) em tarefas de compilação;
- é fundamental que exista um simulador que permita testar (de preferência passo a passo) o código gerado;

Introdução

- Neste momento está funcional um versão escrita em JAVA e pode ser usada, mas tem alguns problemas de implementação, nomeadamente o elevado tempo de execução;

- Neste projeto pretende-se que os alunos reconstruam o simulador com interface interativo e visual para a máquina VM desde o início, em Java ou em C (conforme preferência do grupo) produzindo um executável que evite os problemas atuais.

Stack Virtual VM



Stack Virtual VM

- **Code Block -** Esta pilha contém o conjunto das instruções do programa. Esta é preenchida no inicio da execução do programa, mantendo-se igual do princípio ao fim da execução do mesmo;
- **Operand Stack -** Esta pilha contém valores guardados no registo da maquina. Estes são todos indexados, podendo ser do tipo inteiro, real e endereço;
- **Heap Stack -** Esta pilha contém cadeias de caracteres (strings), cada uma destas referenciado por endereços, cada bloco, contendo um certo numero de caracteres;
- **Call Stack -** Esta pilha contém blocos estruturados, cada um destes referenciado por endereços, cada bloco estruturado contém um certo numero de valores (estes são do mesmo tipo dos que se podem encontrar na Operand Stack);

Stack Virtual VM

- **Stack Pointer -** o registo aponta para o topo corrente da pilha/ para a primeira célula livre da pilha;
- **Frame Pointer -** o registo aponta para o endereço de base das variáveis locais;
- **Global Pointer -** o registo contém o endereço de base das variáveis globais;
- **Program Counter -** o registo aponta para a instrução corrente (da zona de código) por executar;

Especificação dos Objectivos

- Ser facilmente/rapidamente instalável e compatível com o maior numero de sistemas possível;
- Escrito em C ou em Java;
- Execução eficiente para programas grandes;
- Ter pelo menos 2 modos de utilização (silencioso / gráfico);
- Leitura de Input por ficheiro;

- Melhorias na interface que acharmos que façam sentido;

Desenvolvimento

- O simulador foi dividido em 3 partes:
  - Parser
  - Simulador
  - Interface Gráfica

Parser

- O parser implementado com as ferramentas FLEX e YACC;
- Responsável por processar o input para que este fique num estado que possa ser digerido e posteriormente ser guardado no Code Block;

```
START
    PUSHI 10
    pushi 7
    ADD // 10+7
WRITEI   // expected:  17
```



```
START
PUSHI 10
PUSHI 7
ADD
WRITEI
```

Simulador

- A implementação do processador é simples, este apenas espera que o utilizador execute uma ação, e quando existe um pedido para executar a próxima instrução, vai buscar a instrução do Code Block, cujo índice é indicado pelo Program Counter, de seguida executa a função que corresponde ao código da instrução, por sua vez esta função manipula as varias componentes da maquina.

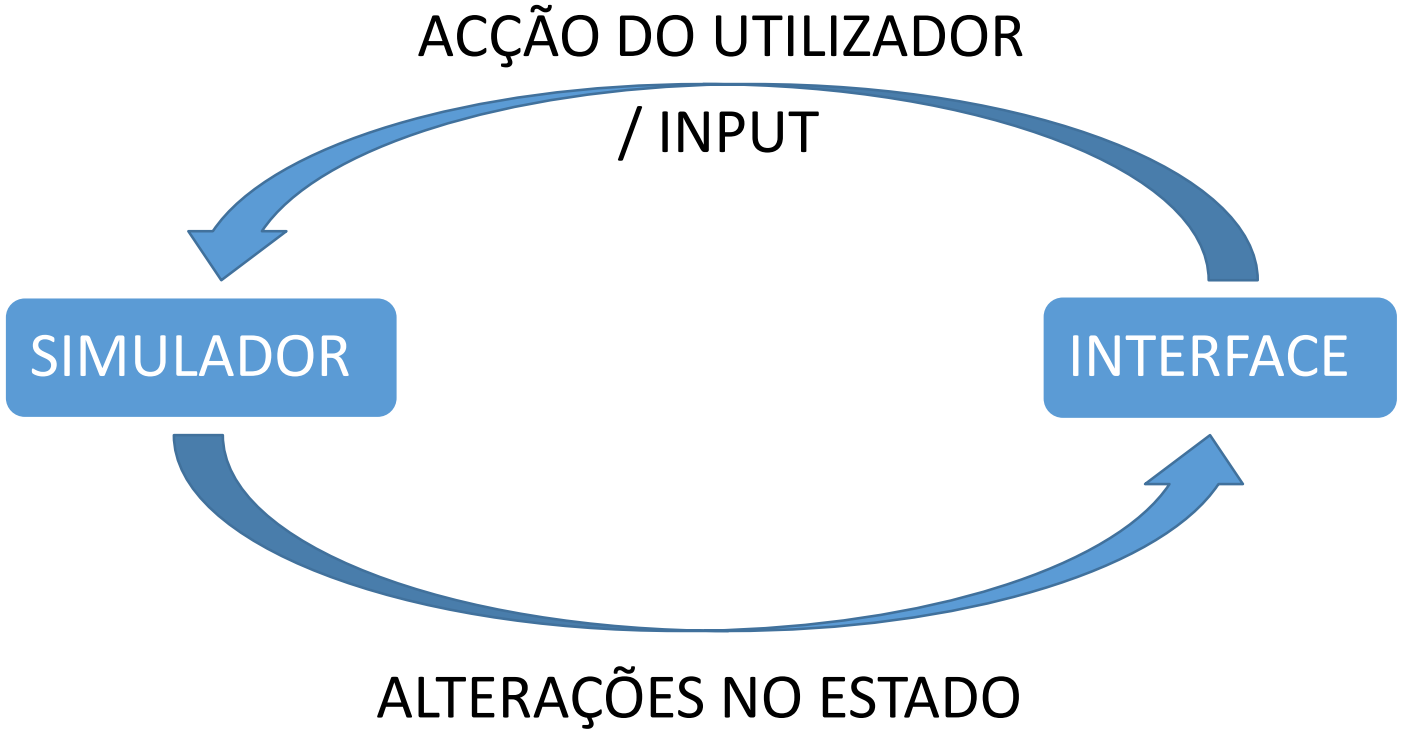
- Ações do utilizador:

  run | next ‘n’ | file ‘f’ | reload | quit

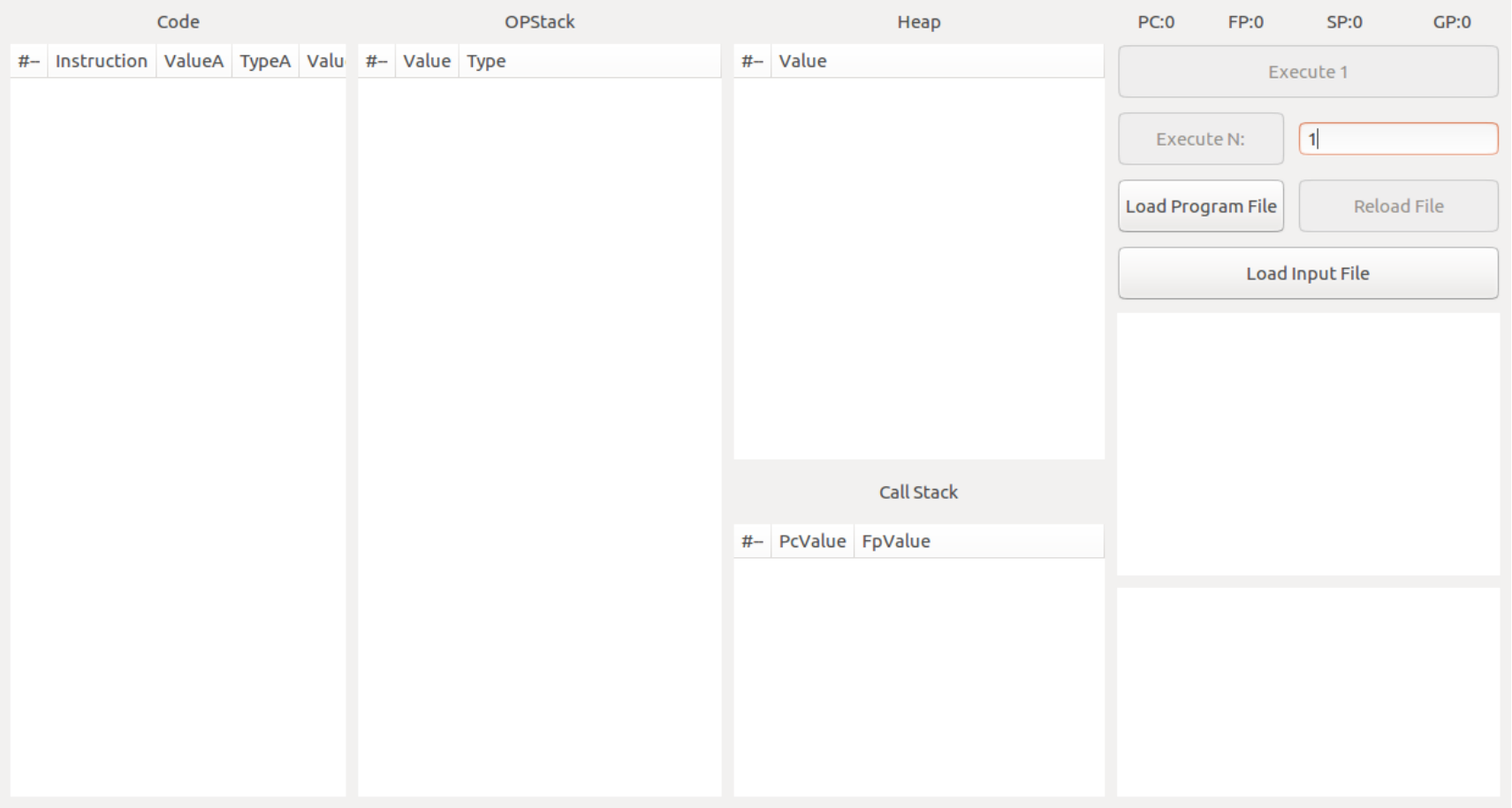
  Input quando necessário para a execução

Interface Gráfica

- Desenvolvido em C com recurso à ferramenta GTK;
- Processo paralelo ao simulador:



Interface Gráfica



Modos de Utilização

- **Máquina em Modo Silencioso –** o programa executado de inicio ao fim, sem existir qualquer partilha de informação do estado da máquina;
- **Máquina em Modo Debug –** o programa executado passo a passo, todas as alterações no estado da máquina são listadas no fim da execução de cada instrução;
- **Máquina em Modo Interface -** o programa executado passo a passo, o estado atual da máquina e apresentado na interface no fim de cada instrução;

Exemplos Testes (Teste 3x9=?)

```
start
    pushi 3
    pushi 9
    mul
    pushs "3 vezes 9 = "
    writes
    writei
    pushs " \n"
    writes
Stop
```

```
> vms 3vezes9.vm
    3 vezes 9 = 27
```

Exemplos Testes (Maior de 2 Números)

```
start
    pushi 0
    pushi 0
    pushs "introduza m:"
    writes
    read
    atoi
    storeg 0
    pushs "introduza n:"
    writes
    read
    atoi
    storeg 1
    pushg 1
```

```
> vms 5-maior2numLidos.vm
    introduza n:-12
    introduza m:14
    O maior e: 14

> vms 5-maior2numLidos.vm
    introduza n:49129
    introduza m:3556
    O maior e: 49129
```

Script Submissões

```
#!/bin/bash

for file in `find . -print | grep -i '.*[.]vm$'`
do
   vms $file < INPUT | diff -y -b - EXP >/dev/null
   if [ $? -eq 0 ]; then
      perl -C -e 'print chr 0x2713'
      echo $file
   else
      perl -C -e 'print chr 0x2717'
      echo $file
   fi
done
```

Demonstrações Live

Conclusão