

TPC6

Respostas aos exercícios propostos

1. **Código em assembly** (com comentários, obtido com `objdump`, que irá também ser usado para os restantes exercícios)

Notas:

- (i) copiar valores da pilha para registos
- (ii) atualizar os valores de x, y e n no corpo do ciclo `do/while`
- (iii) implementação do ciclo `do/while`
- (iv) chamada/regresso da função e gestão da pilha.
- (v) calcular o valor de retorno da função

08048310 <while_loop>:		Nota	Comentário
8048310: 55	push %ebp	(iv)	Salvaguarda %ebp
8048311: 89 e5	mov %esp,%ebp	(iv)	Coloca %ebp igual a %esp (base da pilha)
8048313: 8b 4d 08	mov 0x8(%ebp),%ecx	(i)	Copia arg x para %ecx
8048316: 8b 45 0c	mov 0xc(%ebp),%eax	(i)	Copia arg y para %eax
8048319: 8b 55 10	mov 0x10(%ebp),%edx	(i)	Copia arg n para %edx
804831c: 01 d1	add %edx,%ecx	(ii)	x += n
804831e: 0f af c2	imul %edx,%eax	(ii)	y *= n
8048321: 4a	dec %edx	(ii)	n--
8048322: 85 d2	test %edx,%edx	(iii)	Flags = %edx & %edx (nota: AND bit a bit)
8048324: 7f f6	jg 804831c	(iii)	Salta para início do ciclo se n > 0 (endereço 804831c)
8048326: 8d 04 08	lea (%eax,%ecx,1),%eax	(v)	%eax = x+y (valor de retorno)
8048329: c9	leave	(iv)	Repõe valor de %esp e de %ebp
804832a: c3	ret	(iv)	Regressa (pop %eip)

- 1.b) ver itens (i) na tabela anterior

Variável	Registo atribuído pelo compilador	Instrução Assembly
x	%ecx	mov 0x8(%ebp),%ecx
y	%eax	mov 0xc(%ebp),%eax
n	%edx	mov 0x10(%ebp),%edx

- 1.c) ver itens (ii) na tabela anterior

Depuração do programa com gdb

1d)

- i) Por exemplo: `return(while_loop(4,2,3));`
- ii) ver endereços assinalados a amarelo na tabela
- iii) Escrever `break *0x8048313` e `break *0x804831c` depois de executar `gdb a.out`. Os valores da tabela podem ser confirmados fazendo `run` e `c` (*continue*) após cada paragem
- iv) Estimativa (nota: é mais fácil preencher primeiro a linha correspondente ao `n`)

Variável	Registo	Break1	Break2	Break3	Break4	Break_
x	%ecx	Lixo	4	4+3=7	7+2=9	
y	%eax	Lixo	2	2*3=6	6*2=12	
n	%edx	Lixo	3	2	1	

- Break1 - no primeiro ponto de paragem (`0x8048313`) os valores ainda não estão definidos.
- Break2 - neste ponto de paragem (`0x804831c`) os valores correspondem aos valores passados pelo `main` à função `while_loop` (4, 2 e 3), pois ainda não executou nenhuma iteração do ciclo.
- Break3 - Neste ponto de paragem (novamente `0x804831c` pois este ponto está dentro do ciclo) executou o `add` e o `imul` usando os valores do `n` da iteração anterior pois o `n` é decrementado depois de fazer os cálculos.
- Break 4 - Novamente `0x804831c`. Este é o último ponto de paragem pois quando `n=1` neste ponto de paragem já não vai repetir novamente o ciclo.

Estimativa do conteúdo da estrutura de ativação (*stack frame*)

Pretende-se preencher 3 tipos de informação:

- i. À esquerda do desenho da *stack*, os endereços do início de algumas "caixas".
- ii. No interior das "caixas" o valor numérico que lá deveria estar (pode ser em hexadecimal)
- iii. À direita das "caixas", uma explicação do valor que se encontra na respetiva "caixa".

Cada "caixa" representa 32 bits, ou seja, 4 células de 1 *byte* cada, em que o conteúdo da célula com menor endereço é o *byte* mais à direita de um valor de 32 bits.

A resolução completa implica uma análise do código gerado pelo `gcc`, do conteúdo de alguns registos entre outros aspetos. Por exemplo, no `gdb` é possível visualizar o código da função `main` usando o comando `disas main`:

```
(gdb) disas main
Dump of assembler code for function main:
0x0804832c <main+0>: push    %ebp
0x0804832d <main+1>: mov     %esp,%ebp
0x0804832f <main+3>: sub     $0x8,%esp
0x08048332 <main+6>: and     $0xfffffffff0,%esp
0x08048335 <main+9>: push    %eax
0x08048336 <main+10>: push    $0x3
0x08048338 <main+12>: push    $0x2
0x0804833a <main+14>: push    $0x4
0x0804833c <main+16>: call    0x8048310 <while_loop>
0x08048341 <main+21>: leave
0x08048342 <main+22>: ret
```

Neste caso, o valor do endereço de regresso armazenado na *stack* é o endereço da instrução na *main* imediatamente a seguir à invocação da função (após a instrução `call`), ou seja **0x08048341**. Este endereço pode variar, dependendo da forma como o código C foi escrito.

Esta é a estimativa do conteúdo da *stack frame* associada à função `while_loop`, quando é invocada com os argumentos sugeridos e antes de se confirmar com o `gdb`:

valor de <code>esp/%ebp</code> →	+-----+				
	+-----+				
	+-----+				
	?? ?? ?? ??				Apontador para moldura/quadro anterior
	08 04 83 41				Endereço de regresso da função
	00 00 00 04				Valor do 1º argumento (sugerido: 4)
	00 00 00 02				Valor do 2º argumento (sugerido: 2)
	00 00 00 03				Valor do 3º argumento (sugerido: 3)

O valor do apontador para o quadro anterior (*frame pointer* da *main* assinalado na figura com `???`) pode ser obtido no `gdb`, parando a execução do código logo na 1ª instrução da função e analisando o conteúdo dos registos.

Preenchimento do conteúdo da estrutura de ativação (*stack frame*)

Os valores concretos podem ser obtidos no primeiro ponto de paragem do exercício 1 (`break *0x8048313`), inspecionando as posições de memória apontadas pelo registo `%esp` ou `%ebp`, com o comando `x/6 $esp`.

A lista completa dos comandos será:

```
[XXXXXX@sc TPC6]$ gdb a.out
(gdb) break *0x8048313
Breakpoint 1 at 0x8048313: file while_loop.c, line 2.
(gdb) run
Starting program: /home/XXXXXX/TPC6/a.out
Breakpoint 1, 0x08048313 in while_loop (x=-1073769104, y=1, n=-1073768988)
...
(gdb) x/6 $esp
0xbfff9538:  0xbfff9558  0x08048341  0x00000004  0x00000002
0xbfff9548:  0x00000003  0x00000001
```

Esta informação permite completar o esquema anterior com valores concretos, nomeadamente, o valor de `%esp/%ebp` é `0xbfff9538`, e o valor armazenado em `Mem[%esp]` é `0xbfff9558`.

0xbfff9538 →					
	bf ff 95 58				Apontador para moldura/quadro anterior
	08 04 83 41				Endereço de regresso da função
	00 00 00 04				Valor do 1º argumento (sugerido: 4)
	00 00 00 02				Valor do 2º argumento (sugerido: 2)
	00 00 00 03				Valor do 3º argumento (sugerido: 3)