Imperial College London
Department of Computing

Computer Systems (113)
Exercises – *Classes*

1. Consider

```
class myclass {

      int [100] vec

      int count (int value, int arraylen) {

            int k
            int cnt

            cnt = 0

            for (k = 0; k < arraylen; k++) {
                  if (this.vec [k] == value) {
                        cnt++
                  }
            }
            return cnt
      }
}

int num                 // assume int's are 32-bit
int answer
myclass myobj
...
answer = myobj.count (num, 50)
```

Assume **ints** are 32-bit.

Using the calling convention given in the lectures:

(a) Give the Pentium declarations for the *global* variables: `num`, `answer` and `myobj`

(b) Provide the calling sequence for the assignment statement:

```
answer = myobj.count (num, 50)
```

(c) Provide a Pentium implementation for method `count` and show the stack frame (activation record) for the call in part b, i.e. the state of the stack before the first statement in the method is executed.

Imperial College London
Department of Computing

Computer Systems (113)

Solutions – *Pentium Programming*

1a) Declarations for the global variables

```
num    resd  1
answer resd  1
myobj  resd  100
```

b) Code sequence for `answer = myobj.count(num, 50)`

```
      push dword 50             ; push 2nd parameter
      push dword [num]          ; push 1st parameter
      push dword myobj          ; push object instance
      call myclass_count        ; call method myclass.count
      add  esp, 12              ; remove parameters & object instance
      mov  [answer], eax        ; copy method result into answer
```

(c) Pentium assembly language implementation for method Count:

Stack Frame (Activation Record)

```
      [ebp+16]     arraylen = 50
      [ebp+12]     value = num
      [ebp+8]      myobj
      [ebp+4]      return address
      [ebp]        caller's ebp
                   saved ebx                ebx will hold value
                   saved ecx                ecx will hold k
      [esp]        saved esi                esi will hold vec
```

```
myclass_count:
      push ebp                  ; save caller's ebp
      mov  ebp, esp             ; set new ebp
      push ebx                  ; ebx will hold value
      push ecx                  ; ecx will hold k
      push esi                  ; esi will hold vec

      mov  ebx, [ebp+12]        ; copy value into ebx
      mov  esi, [ebp+8]         ; copy object instance (i.e. this) into esi

      mov  eax, 0               ; eax will hold cnt
      mov  ecx, 0               ; k = 0
next:
      cmp  ecx, [ebp+16]        ; compare k with arraylen
      jge  endfor               ; jump if k >= arraylen

      cmp  [esi+4*ecx], ebx     ; compare vec [k] with value.
      jne  endif                ; if not equal skip to end of if statement
      inc  eax                  ; cnt++
endif:
      inc  ecx                  ; k++
      jmp  next
endfor:
      pop  esi                  ; restore esi
      pop  ecx                  ; restore ecx
      pop  ebx                  ; restore ebx
      pop  ebp                  ; restore ebp
      ret
```