```
#*****************************************************************************
# File: es1.s
#       Contains the Assembly translation for es1.cpp.
#
# Author: Rambod Rahmani <rambodrahmani@autistici.org>
#         Created on 14/09/2019.
#*****************************************************************************

#-----------------------------------------------------------------------------
.TEXT
.GLOBAL _ZN2clC1EPc                                          # cl::cl(char v[])
#-----------------------------------------------------------------------------
# activation record:
# ------------------
#  i              -20
#  &v             -16
#  this           -8
#  %rbp           0
#-----------------------------------------------------------------------------
_ZN2clC1EPc:
# set stack locations labels
    .set this, -8
    .set v,    -16
    .set i,    -20

# prologue: activation frame
    pushq %rbp
    movq  %rsp, %rbp
    subq  $24, %rsp                  # reserve stack space for actual arguments

# copy actual arguments to the stack
    movq %rdi, this(%rbp)
    movq %rsi, v(%rbp)

# a = v[0]++;
    movl   $0, i(%rbp)               # i = 0
    movslq i(%rbp), %rcx             # i -> %rcx
    movb   (%rsi, %rcx, 1), %al      # v[0] -> %al
    movb   %al, (%rdi, %rcx, 1)      # a = v[0];
    incb   (%rsi, %rcx, 1)           # v[0]++

# b = v[1];
    incq %rcx
    movb (%rsi, %rcx, 1), %al        # v[1] -> %al
    movb %al, (%rdi, %rcx, 1)        # b = v[1];

# for loop initialization
    movl $0, i(%rbp)                 # i = 0

for:
    cmpl $4, i(%rbp)                 # check if i < 4
    jge  finefor                     # end for loop (i >= 4)

# for loop body
    movq   this(%rbp), %rdi          # this -> %rdi
    movq   v(%rbp), %rsi             # &v -> %rsi
    movslq i(%rbp), %rcx             # i -> %rcx

    movq $0, %r8                     # $0 -> %r8
    movb (%rdi, %r8, 1), %al         # a -> %al

    incq %r8                         # increment %r8
    movb (%rdi, %r8, 1), %bl         # b -> %bl

    addb (%rsi, %rcx, 1), %al        # v[i] + a -> %al
    movb %al, 8(%rdi, %rcx, 1)       # s.vv1[i] = v[i] + a, [0]

    addb   (%rsi, %rcx, 1), %bl      # v[i] + b -> %bl
    movsbq %bl, %rbx                 # %bl -> %rbx
    movq   %rbx, 16(%rdi, %rcx, 8)   # s.vv2[i] = v[i] + b;
```

```
    incl i(%rbp)
    jmp  for

finefor:

    movq  this(%rbp), %rax            # return initialized object address
    leave                             # movq %rbp, %rsp; popq %rbp
    ret

#-----------------------------------------------------------------------------
.GLOBAL _ZN2cl5elab1ER2sti
#-----------------------------------------------------------------------------
# activation record:
# ------------------
#  i             -24
#  d             -20
#  &ss           -16
#  this          -8
#  %rbp          0
#-----------------------------------------------------------------------------
_ZN2cl5elab1ER2sti:
# set stack locations labels
    .set this, -8
    .set ss,   -16
    .set d,    -20
    .set i,    -24

# prologue: activation frame
    pushq %rbp
    movq  %rsp, %rbp
    subq  $24, %rsp                 # reserve stack space for actual arguments

# copy actual arguments to the stack
    movq %rdi, this(%rbp)
    movq %rsi, ss(%rbp)
    movl %edx, d(%rbp)

# for loop initialization
    movl $0, i(%rbp)                # i = 0

for1:
    cmpl $4, i(%rbp)                # check if i < 4
    jge  finefor1                   # end for loop (i >= 4)

# for looop body
    movq   this(%rbp), %rdi
    movq   ss(%rbp), %rsi
    movslq i(%rbp), %rcx

# if (d >= ss.vv2[i])
    movq 8(%rsi, %rcx, 8), %rax     # ss.vv2[i] -> %rax
    cmpl %eax, d(%rbp)              # d >= ss.vv2[i]
    jl   fineif                     # exit if (d < ss.vv2[i])
    movb (%rsi, %rcx, 1), %al       # ss.vv1[i] -> %al
    addb %al, 8(%rdi, %rcx, 1)      # s.vv1[i] += ss.vv1[i];

fineif:
    movq   $0, %r8                  # 0 -> %r8
    movb   (%rdi, %r8, 1), %al      # a -> %al
    movsbl %al, %eax                # %al => %eax
    addl   d(%rbp), %eax            # d + a -> %eax
    movslq %eax, %rax               # %eax => %rax
    movq   %rax, 16(%rdi, %rcx, 8)  # s.vv2[i] = a + d;

    incl i(%rbp)
    jmp  for1

finefor1:
```

```
    leave                           # movq %rsp, %rbp; popq %rbp
    ret

#*****************************************************************************

#############################################################################
# [0]
# When it comes to structs memory alignment the following criteria is applied:
# "In general, a struct instance will have the alignment of its widest scalar
#  member. Compilers do this as the easiest way to ensure that all the members
#  are self-aligned for fast access".
# In this case the struct will have 8-byte alignment because it contains a field
# of type long.
#
# Primitive data types typical alignments:
#  - A char (one byte) will be 1-byte aligned.
#  - A short (two bytes) will be 2-byte aligned.
#  - An int (four bytes) will be 4-byte aligned.
#  - A long (eight bytes) will be 8-byte aligned.
#  - Any pointer (eight bytes) will be 8-byte aligned.
#############################################################################
```