```
/**
 * File: cc.h
 *       Contains the declaration for the data structures used in the exercise.
 *
 * Author: Rambod Rahmani <rambodrahmani@autistici.org>
 *         Created on 17/07/2019.
 */

#include <iostream>

using namespace std;

struct st1
{
    char vc[4];
};

struct st2
{
    int vd[4];
};

class cl
{
    st1 s;
    long v[4];

public:
    cl(char c, st2 s2);

    void elab1(st1& s1, st2& s2);

    void stampa()
    {
        int i;

        for (i = 0; i < 4; i++)
        {
            cout << s.vc[i] << ' ';
        }
        cout << endl;

        for (i = 0; i < 4; i++)
        {
            cout << v[i] << ' ';
        }
        cout << endl << endl;
    }
};
```

```cpp
/**
 * File: es1.cpp
 *       Contains the C++ code to be translated into Assembly (es1.s file).
 *
 * Author: Rambod Rahmani <rambodrahmani@autistici.org>
 *         Created on 17/07/2019.
 */

#include "cc.h"

cl::cl(char c, st2 s2)
{
    for (int i = 0; i < 4; i++)
    {
        s.vc[i] = c;

        v[i] = s2.vd[i] + s.vc[i];
    }
}

void cl::elab1(st1 & s1, st2 & s2)
{
    cl cla('f', s2);

    for (int i = 0; i < 4; i++)
    {
        if (s.vc[i] < s1.vc[i])
        {
            s.vc[i] = cla.s.vc[i];
        }

        if (v[i] < cla.v[i])
        {
            v[i] += cla.v[i];
        }
    }
}
```

```
a a a a
98 117 100 137

f f f f
210 117 232 137
```

```
#*******************************************************************************
# File: es1.s
#       Contains the Assembly translation for es1.cpp.
#
# Author: Rambod Rahmani <rambodrahmani@autistici.org>
#         Created on 17/07/2019.
#*******************************************************************************

#-------------------------------------------------------------------------------
.TEXT
.GLOBAL _ZN2clC1Ec3st2                              # cl::cl(char c, st2 s2)
#-------------------------------------------------------------------------------
# activation record:
# ------------------
#  s2 LSB      -32 -------------> s2.vd[0]   -32
#  s2 MSB      -24                s2.vd[1]   -28
#  c           -16                s2.vd[2]   -24
#  i           -12                s2.vd[3]   -20
#  this        -8
#  %rbp        0
#-------------------------------------------------------------------------------
_ZN2clC1Ec3st2:
    pushq  %rbp                         # prologue
    movq   %rsp, %rbp
    subq   $32,  %rsp                   # reserve stack space for actual arguments

# copy actual arguments to the stack, [1]
    movq   %rdi, -8(%rbp)               # this
    movb   %sil, -16(%rbp)              # c
    movq   %rdx, -32(%rbp)              # s2 (LSB)
    movq   %rcx, -24(%rbp)              # s2 (MSB)

# for loop, initialization
    movl   $0, -12(%rbp)                # i = 0

for:
    cmpl   $4, -12(%rbp)                # i < 4
    jge    finefor                      # end loop (i >= 4)

# for loop body
    movq    -8(%rbp),  %rdi             # this -> %rdi
    movslq -12(%rbp), %rcx              # i -> %rcx
    movb    -16(%rbp), %al              # c -> %al
    movb    %al, (%rdi, %rcx)           # %al -> s.vc[i]

    movsbl (%rdi, %rcx), %eax           # s.vc[i] -> %eax
    addl    -32(%rbp, %rcx, 4), %eax    # s2.vd[i] + s.vc[i] -> %eax
    movslq %eax, %rax                   # sign extend to 64-bits
    movq    %rax, 8(%rdi, %rcx, 8)      # v[i] = s2.vd[i] + s.vc[i];

    incl    -12(%rbp)                   # i++
    jmp     for                         # loop again

finefor:
    movq    -8(%rbp), %rax              # return initialized object address
    leave                               # mov %rbp, %rsp; pop %rbp
    ret

#-------------------------------------------------------------------------------
.GLOBAL _ZN2cl5elab1ER3st1R3st2            # void cl::elab1(st1 & s1, st2 & s2)
#-------------------------------------------------------------------------------
# activation record:
# ------------------
#  i           -72
#  cla         -64
#  cla.v[0]    -56
#  cla.v[1]    -48
#  cla.v[2]    -40
#  cla.v[3]    -32
#  s2          -24
#  s1          -16
#  this        -8
#  %rbp        0
```

```
#--------------------------------------------------------------------------------
_ZN2cl5elab1ER3st1R3st2:
    pushq  %rbp                     # prologue
    movq   %rsp, %rbp
    subq   $72, %rsp                # reserve stack space for actual arguments

# copy actual arguments to the stack
    movq  %rdi, -8(%rbp)            # this
    movq  %rsi, -16(%rbp)           # s1
    movq  %rdx, -24(%rbp)           # s2

# cl cla('f', s2);
    leaq  -64(%rbp), %rdi           # &cla (this -> %rdi)
    movb  $'f', %sil                # c -> %rsi
    movq  -24(%rbp), %r8            # &s2 -> %r8
    movq  (%r8), %rdx               # &s2 -> %rdx (LSB)
    movq  8(%r8), %rcx              # &s2 -> %rdx (MSB)
    call  _ZN2clC1Ec3st2            # call constructor

# for loop, initialization
    movl  $0, -72(%rbp)             # i = 0

for2:
    cmpl  $4, -72(%rbp)             # i < 4
    jge   finefor2                  # end loop (i >= 4)

# for loop body
    movq   -8(%rbp), %rdi           # this -> %rdi
    movslq -72(%rbp), %rcx          # i -> %rcx
    movb   (%rdi, %rcx), %al        # s.vc[i] -> %al
    movq   -16(%rbp), %r8           # s1 -> %r8
    cmpb   %al, (%r8, %rcx)         #
    jle    fineif                   # if (s1.vc[i] <= s.vc[i]) jump

# if (s.vc[i] < s1.vc[i])
    movb  -64(%rbp, %rcx), %al      # cla.s.vc[i] -> %al
    movb  %al, (%rdi, %rcx)         # %al -> s.vc[i]

fineif:

    movq  8(%rdi, %rcx, 8), %rax    # s.v[i] -> %rax
    cmpq  %rax, -56(%rbp, %rcx, 8)  #
    jle   fineif2                   # if (cla.v[i] <= v[i]) jump

# if (v[i] < cla.v[i])
    movq -56(%rbp, %rcx, 8), %rax   # cla.v[i] -> %rax
    addq %rax, 8(%rdi, %rcx, 8)     # v[i] += cla.v[i];

fineif2:

    incl  -72(%rbp)                 # i++
    jmp   for2                      # loop again

finefor2:
    leave                           # mov %rbp, %rsp; pop %rbp
    ret
#********************************************************************************

#********************************************************************************
# [1]
# On x86-64 UNIX systems, including Linux and default NetRun, the first six
# parameters go into rdi, rsi, rdx, rcx, r8, and r9.

#********************************************************************************
# Curiously, you can write a 64-bit value into rax, then read off the low 32
# bits from eax, or the low 16 bitx from ax, or the low 8 bits from al - it's
# just one register, but they keep on extending it!
#
# ------------------------------------------------------------------------------
# | rax: 64-bit                          | eax: 32-bit      | ax: 16-bit    |ah|al|
# ------------------------------------------------------------------------------

#********************************************************************************
```

```
# C++ Storage Sizes
# -----------------
# Object    C++ Name    Bits    Bytes (8 bits)
# --------------------------------------------
# Bit       None        1       < 1
# Byte      char        8       1
# WORD      short       16      2
# DWORD     int         32      4
# QWORD     long        64      8
# --------------------------------------------
```

```
/**
 * File: prova1.cpp
 *       This file contains a developer harness test for es1.s.
 *
 *       Compile with:
 *          g++ -o es1 -fno-elide-constructors es1.s prova1.cpp
 *
 *       Test your result with:
 *          ./es1 | diff - es1.out
 *
 * Author: Rambod Rahmani <rambodrahmani@autistici.org>
 *         Created on 17/07/2019.
 */

#include "cc.h"

/**
 * Developer harness test.
 *
 * @param  argc  command line arguments counter.
 * @param  argv  command line arguments.
 *
 * @return       execution exit code.
 */
int main(int argc, char * argv[])
{
    st1 s1 = { 'e', 'b', 'f', 'd' };

    st2 sa = { 1, 20, 3, 40 };

    st2 sb = { 10, 2, 30, 4 };

    cl cla('a', sa);

    cla.stampa();

    cla.elab1(s1, sb);

    cla.stampa();

    return 0;
}
```