Practical 1

Software Reengineering • Auburn University • COMP 7720/7726/4970 • Summer 2018

Below is a small C program, together with an assembly language listing generated by Visual C++ (with optimizations disabled). Note the line numbers in the assembly language code.

```c
#include<stdio.h>

enum Mode {upper, lower};
char arr[] = {'h','E','L','L','O',' ','w','O','R','L','D','\0'};

int __fastcall uplow_lowup(char* letters, int size, enum Mode m ){
    int i = 0;
    int count = 0;

    while( i < size){
        if(letters[i] >= 65 && letters[i] <= 90){ // A = 65, Z = 90

            letters[i] = letters[i] + 32;
            if(m == lower){
                count ++;

        }else if (letters[i] >= 97 && letters[i] <= 122){

            letters[i] = letters[i] - 32;
            if(m == upper)
                count ++;
        }
        i++;
    }
    return count;
}
int main(){
    printf("%d: %s\n", uplow_lowup(arr, 11, upper), arr);
    return 0;
}
```

```
1   ; Listing generated by Microsoft (R) Optimizing Compiler Version 16.00.30319.01
2       TITLE   C:\Users\hkofa\documents\visual studio 2010\...\v1.c
3       .686P
4       .XMM
5       include listing.inc
6       .model  flat
7   INCLUDELIB OLDNAMES
8   PUBLIC  _arr
9   EXTRN   @__security_check_cookie@4:PROC
10  EXTRN   __imp__printf:PROC
11  _arr    DB  068H
12      DB  045H
13      DB  04cH
14      DB  04cH
15      DB  04fH
16      DB  020H
17      DB  077H
18      DB  04fH
19      DB  052H
20      DB  04cH
21      DB  044H
22      DB  00H
```

```
$SG-5   DB  '%d: %s', 0aH, 00H
PUBLIC  @uplow_lowup@12
; Function compile flags: /Odtp
;   COMDAT @uplow_lowup@12
_TEXT   SEGMENT
_size$ = -16                        ; size = 4
_letters$ = -12                     ; size = 4
_i$ = -8                        ; size = 4
_count$ = -4                        ; size = 4
_m$ = 8                         ; size = 4
@uplow_lowup@12 PROC                    ; COMDAT
; _letters$ = ecx
; _size$ = edx
; File c:\users\hkofa\documents\visual studio 2010\projects\practical1\practical1\v1.c
    push    ebp
    mov ebp, esp
    sub esp, 16                 ; 00000010H
    mov DWORD PTR _size$[ebp], edx
    mov DWORD PTR _letters$[ebp], ecx

    mov DWORD PTR _i$[ebp], 0

    mov DWORD PTR _count$[ebp], 0
$LN7@:

    mov eax, DWORD PTR _i$[ebp]
    cmp eax, DWORD PTR _size$[ebp]
    jge $LN6@

    mov ecx, DWORD PTR _letters$[ebp]
    add ecx, DWORD PTR _i$[ebp]
    movsx   edx, BYTE PTR [ecx]
    cmp edx, 65                 ; 00000041H
    jl  SHORT $LN5@
    mov eax, DWORD PTR _letters$[ebp]
    add eax, DWORD PTR _i$[ebp]
    movsx   ecx, BYTE PTR [eax]
    cmp ecx, 90                 ; 0000005aH
    jg  SHORT $LN5@

    mov edx, DWORD PTR _letters$[ebp]
    add edx, DWORD PTR _i$[ebp]
    movsx   eax, BYTE PTR [edx]
    add eax, 32                 ; 00000020H
    mov ecx, DWORD PTR _letters$[ebp]
    add ecx, DWORD PTR _i$[ebp]
    mov BYTE PTR [ecx], al

    cmp DWORD PTR _m$[ebp], 1
    jne SHORT $LN4@

    mov edx, DWORD PTR _count$[ebp]
    add edx, 1
    mov DWORD PTR _count$[ebp], edx
$LN4@:

    jmp SHORT $LN3@
$LN5@:

    mov eax, DWORD PTR _letters$[ebp]
    add eax, DWORD PTR _i$[ebp]
    movsx   ecx, BYTE PTR [eax]
    cmp ecx, 97                 ; 00000061H
```

```
86      jl  SHORT $LN3@
87      mov edx, DWORD PTR _letters$[ebp]
88      add edx, DWORD PTR _i$[ebp]
89      movsx   eax, BYTE PTR [edx]
90      cmp eax, 122                ; 0000007aH
91      jg  SHORT $LN3@
92
93      mov ecx, DWORD PTR _letters$[ebp]
94      add ecx, DWORD PTR _i$[ebp]
95      movsx   edx, BYTE PTR [ecx]
96      sub edx, 32                 ; 00000020H
97      mov eax, DWORD PTR _letters$[ebp]
98      add eax, DWORD PTR _i$[ebp]
99      mov BYTE PTR [eax], dl
100
101     cmp DWORD PTR _m$[ebp], 0
102     jne SHORT $LN3@
103
104     mov ecx, DWORD PTR _count$[ebp]
105     add ecx, 1
106     mov DWORD PTR _count$[ebp], ecx
107 $LN3@:
108
109     mov edx, DWORD PTR _i$[ebp]
110     add edx, 1
111     mov DWORD PTR _i$[ebp], edx
112
113     jmp $LN7@
114 $LN6@:
115
116     mov eax, DWORD PTR _count$[ebp]
117
118     mov esp, ebp
119     pop ebp
120     ret 4
121 @uplow_lowup@12 ENDP
122 _TEXT   ENDS
123 PUBLIC  _main
124 ; Function compile flags: /Odtp
125 ;   COMDAT _main
126 _TEXT   SEGMENT
127 _main   PROC                        ; COMDAT
128
129     push    ebp
130     mov ebp, esp
131
132     push    0
133     mov edx, 11                 ; 0000000bH
134     mov ecx, OFFSET _arr
135     call    @uplow_lowup@12
136     push    OFFSET _arr
137     push    eax
138     push    OFFSET $SG-5
139     call    DWORD PTR __imp__printf
140     add esp, 12                 ; 0000000cH
141
142     xor eax, eax
143
144     pop ebp
145     ret 0
146 _main   ENDP
147 _TEXT   ENDS
148 END
```

Q1. Based on the c & assembly code answer the following questions:

a. briefly, describe what does `uplow_lowup` function do?
b. What is the call convention used (by `uplow_lowup`), and which lines of the assembly code indicate that?
c. How many parameter(s) is/are being passed to `uplow_lowup` via the stack?
d. How many local variables are being used by `uplow_lowup`? list the assembly lines that support your answer.
e. Describe what will be the content of the ecx register after executing line number 52 for the first time? (not asking bout actual value)
f. Explain the purpose of the following lines: 48,49,50?
g. Which register will contain the return value of `uplow_lowup` fucntion, and which line sets the final value into that register?
h. Which was the first argument pushed into the stack, and where was it used in the assembly code?
i. Would you consider the assembly code mentioned above as an optimal implementation for the c code? Why/why not?

Q2. Suppose the following changes (marked in bold) were made to the C code above.

```
#include<stdio.h>
enum Mode {upper, lower} ;
char arr[] = {'h','E','L','L','O',' ','w','O','R','L','D','\0'};


unsigned short __cdecl uplow_lowup(char* letters, int size, enum Mode m ){
    int i = 0;
    unsigned short count = 0;

    while( i < size){
        if(letters[i] >= 65 && letters[i] <= 90){ // A = 65, Z = 90
            letters[i] = letters[i] + 32;
            if(m == lower)
                count ++;

        }else if (letters[i] >= 97 && letters[i] <= 122){
            letters[i] = letters[i] - 32;
            if(m == upper)
                count ++;
        }
        i++;
    }
    return count;
}
int main(){

    printf("%u: %s\n", uplow_lowup(arr, 11, upper), arr);
    return 0;
}
```

Which of the following lines of the assembly language code would change? (pick all that apply, and briefly describe why).

a. Lines 11-22 (DB directives).
b. Line 23 (DD directive).
c. Lines 24, 33, 121, 135 (function name).
d. Lines 28-32 (offsets for stack content).
e. Lines 37-38 (stack frame creation).
f. Line 39 (esp register).
g. Lines 40-41 (dealing with args).
h. Lines 48-50.
i. Lines 74-76 (changing the count value).
j. Lines 93-96 (toupper).
k. Line 116 (seting eax value).
l. Lines 118-119 (terminate stack frame) .
m. Line 120 (ret 4).
n. Lines 129-130 (create stack frame).
o. Lines 132-134 (call _uplow_lowup from main).
p. Lines 139, 140.