# Practical 3

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

The last page of this PDF contains IDA's display of RtlEqualString & RtlUpperString, two extra functions from ntdll.dll. Your job is to reverse engineer these functions (+ one more function for the Grad) and answer some questions about its code.

If you Google "RtlEqualString ntdll," the first result you get will be a man page from the Wine project (recall that Wine is a Windows emulation layer for Linux).

## NAME

**RtlEqualString** *(NTDLL.@)*

## SYNOPSIS

```
BOOLEAN RtlEqualString
(
 const STRING* s1,
 const STRING* s2,
 BOOLEAN       CaseInsensitive
)
```

## DESCRIPTION

Determine if two strings are equal.

## PARAMS

| | |
|---|---|
| *s1* | *[In]* Source string. |
| *s2* | *[In]* String to compare to *s1*. |
| *CaseInsensitive* | *[In]* TRUE = Case insensitive, FALSE = Case sensitive. |

## RETURNS

Non-zero if *s1* is equal to *s2*, 0 otherwise.

STRING is a struct type:

```
typedef struct {
     unsigned short Length;
     unsigned short MaximumLength;
     char *Buffer;
} STRING;
```

(continued next page)

This is the documentation page for RtlUpperString from Wine project.

## NAME

**RtlUpperString** *(NTDLL.@)*

## SYNOPSIS

```
void RtlUpperString
(
 STRING*       dst,
 const STRING* src
)
```

## DESCRIPTION

Converts an Ascii string to uppercase.

## PARAMS

*dst* *[Out]* Destination for converted string.
*src* *[In]* Source string to convert.

## RETURNS

Nothing.

(continued next page)

Wine aims to be an API-compatible with Windows, and it is quite successful. Thus, it is fine to get the above information from Wine's documentation, because we are only looking at the interface they provide. However...

**Do not look at Wine's source code implementing the RtlUpperString & RtlEqualString function.**

Why? The assembly code you are reversing is from Microsoft's version of ntdll.dll. Wine provides the same API, but it was completely re-implemented. You should not expect Wine's code to be the same as Microsoft's.

As you are reversing RtlEqualString & RtlUpperString, you will see that it calls RtlUpperChar. Here is Wine's documentation for that function:

## NAME

**RtlUpperChar** *(NTDLL.@)*

## SYNOPSIS

```
CHAR RtlUpperChar
(
 CHAR ch
)
```

## DESCRIPTION

Converts an Ascii character to uppercase.

## PARAMS

*ch [In]* Character to convert.

## RETURNS

The uppercase character value.

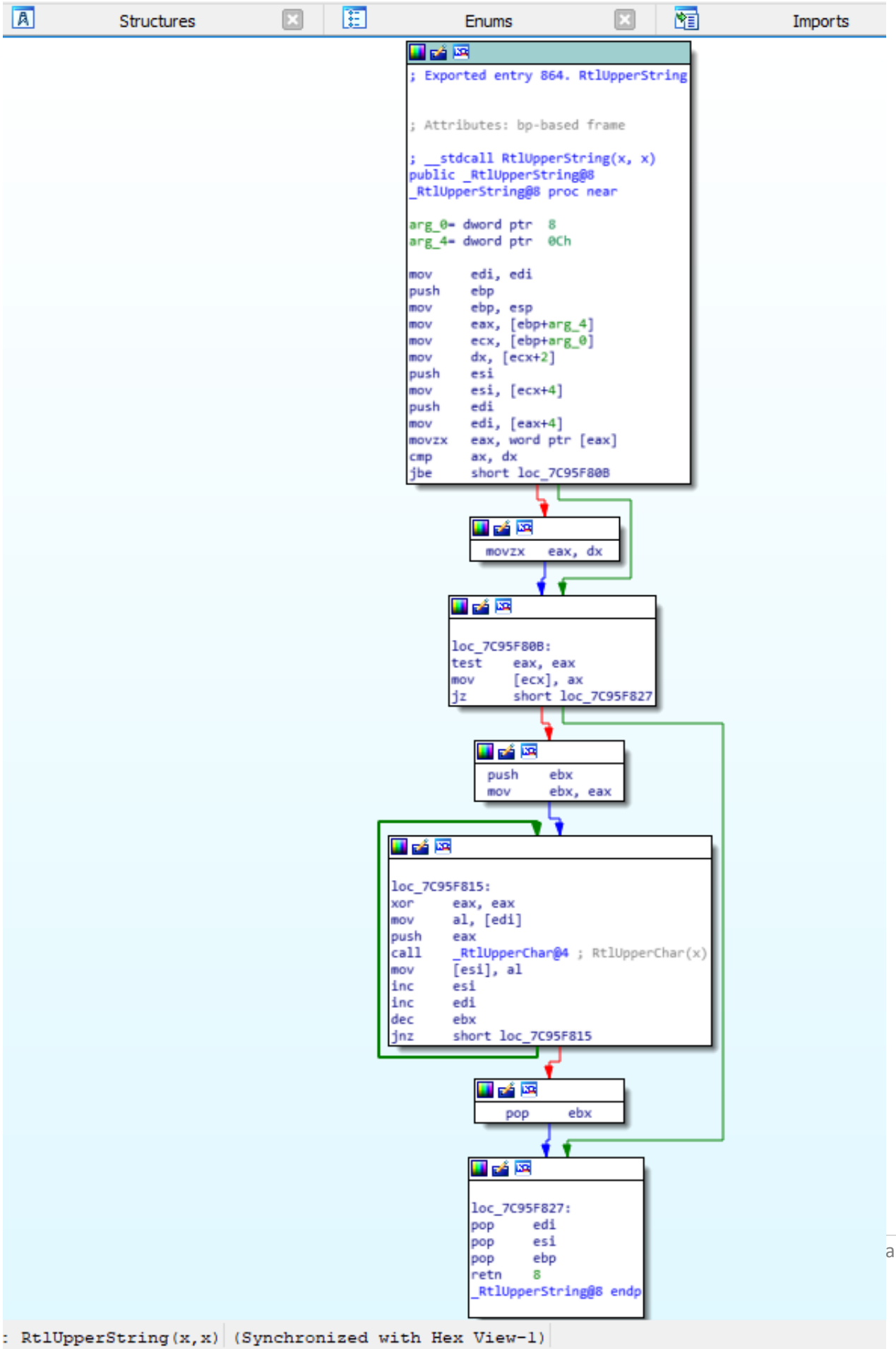
(continued next page)

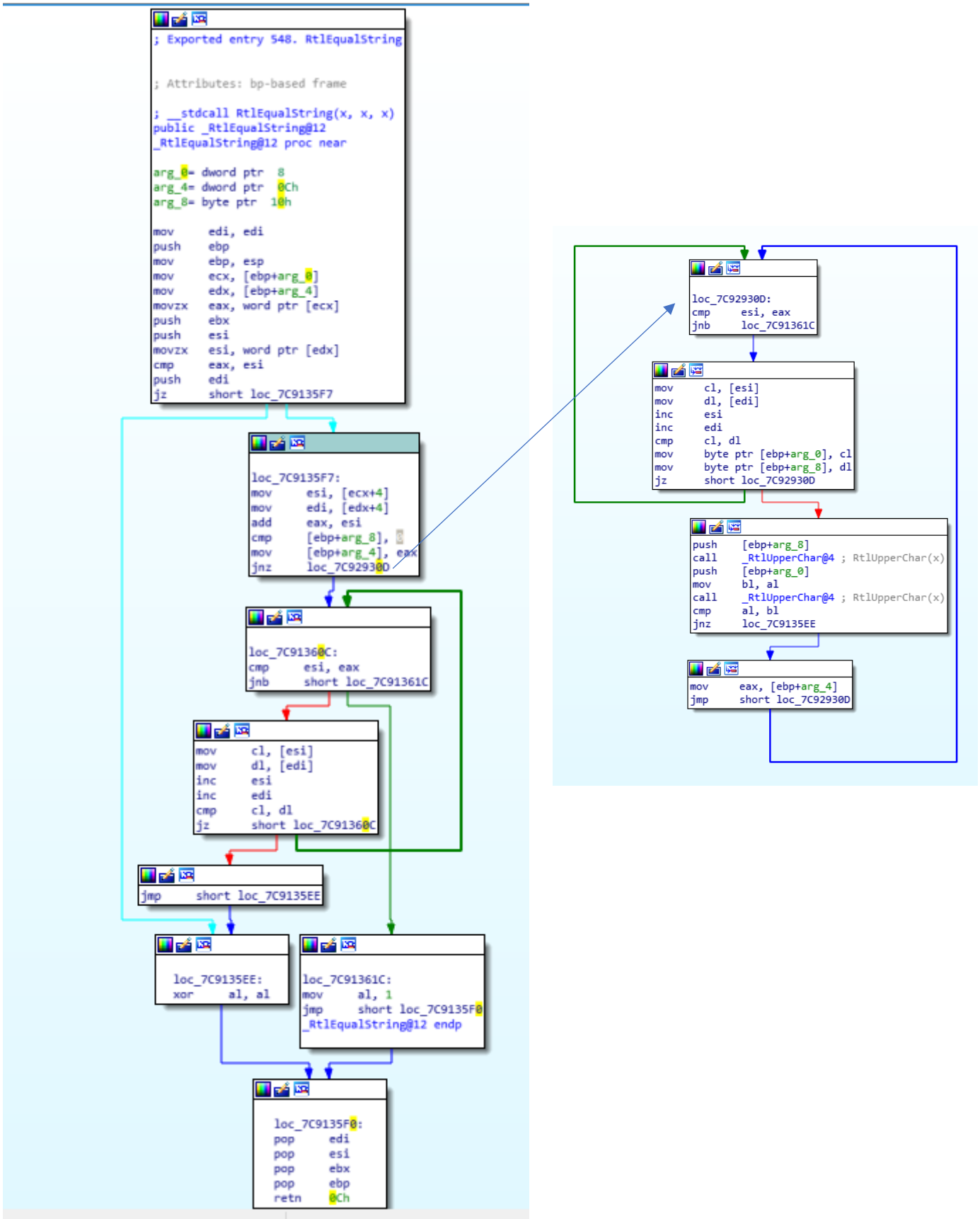
# Questions

**Note1:** *In your answers don't use plain register names, for example if edx is zero then jump into line 0x00400... Your answer should describe the actual content of edx and where the jump will be and why we took that jump in the first place.*
**Note2:** *Question 7 & 10 will have a higher grading load.*

Q1. IDA detected that RtlUpperString & RtlEqualString are __stdcall functions with two and three arguments, respectively. What clues in the assembly code tell you that this is the case?
Q2. RtlUpperString: at line .text:7C95F803 the cmp instraction compare **ax** with **dx**, is it a sigend or unsiged comparison and why?
Q3. RtlUpperString: why movzx instraction was used rather than regular mov at lines .text:7C95F800 & .text:7C95F808?
Q4. RtlUpperString: describe what will be the content of **ax** and **dx** registers when executing the cmp instraction at line .text:7C95F803?
Q5. RtlUpperString: Under what circumstances will the jump to loc_7C95F827 be taken?
Q6. RtlUpperString: the location loc_7C95F815 is a strat of a loop, in english describe what is happening within that loop? And when it will terminate?
Q7. RtlUpperString: If we call **RtlUpperString(str1, str2)**, what will be the value of str1 (mention all members: length, MaximumLength and Buffer) after we return from the funtion call? Where STRING ***str1****{length=6, MaximumLength=8, Buffer="Hello!"}*, ***str2****{ length=10, MaximumLength =30, Buffer ="Good Bye!!"}*.
Q8. RtlEqualString: Under what circumstances will the jump to loc_7C92930D be taken (hint: more than one)?
Q9. [**Grad only**, bonus for undergrad] RtlEqualString: decribe what is the differnce between the cmp instractions at lines .text:7C913616, .text:7C92931B and .text:7C929337?
Q10. [**Grad only**, bonus for undergrad] RtlUpperChar: based on arg1 there are three main cases that will control the branching within the code, in english mention those cases. Note: *no need to talk in details when refering to loc_7C94C431*.

Submit your answers in a text, word or pdf document to canvas.


Structures
Enums
Imports
; Exported entry 864. RtlUpperString
; Attributes: bp-based frame
; __stdcall RtlUpperString(x, x)
public _RtlUpperString@8
_RtlUpperString@8 proc near
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
mov edi, edi
push ebp
mov ebp, esp
mov eax, [ebp+arg_4]
mov ecx, [ebp+arg_0]
mov dx, [ecx+2]
push esi
mov esi, [ecx+4]
push edi
mov edi, [eax+4]
movzx eax, word ptr [eax]
cmp ax, dx
jbe short loc_7C95F80B
movzx eax, dx
loc_7C95F80B:
test eax, eax
mov [ecx], ax
jz short loc_7C95F827
push ebx
mov ebx, eax
loc_7C95F815:
xor eax, eax
mov al, [edi]
push eax
call _RtlUpperChar@4 ; RtlUpperChar(x)
mov [esi], al
inc esi
inc edi
dec ebx
jnz short loc_7C95F815
pop ebx
loc_7C95F827:
pop edi
pop esi
pop ebp
retn 8
_RtlUpperString@8 endp
: RtlUpperString(x,x) (Synchronized with Hex View-1)


; Exported entry 548. RtlEqualString
; Attributes: bp-based frame
; __stdcall RtlEqualString(x, x, x)
public _RtlEqualString@12
_RtlEqualString@12 proc near
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
arg_8= byte ptr 10h
mov edi, edi
push ebp
mov ebp, esp
mov ecx, [ebp+arg_0]
mov edx, [ebp+arg_4]
movzx eax, word ptr [ecx]
push ebx
push esi
movzx esi, word ptr [edx]
cmp eax, esi
push edi
jz short loc_7C9135F7
loc_7C9135F7:
mov esi, [ecx+4]
mov edi, [edx+4]
add eax, esi
cmp [ebp+arg_8], 0
mov [ebp+arg_4], eax
jnz loc_7C92930D
loc_7C92930D:
cmp esi, eax
jnb loc_7C91361C
mov cl, [esi]
mov dl, [edi]
inc esi
inc edi
cmp cl, dl
mov byte ptr [ebp+arg_0], cl
mov byte ptr [ebp+arg_8], dl
jz short loc_7C92930D
push [ebp+arg_8]
call _RtlUpperChar@4 ; RtlUpperChar(x)
push [ebp+arg_0]
mov bl, al
call _RtlUpperChar@4 ; RtlUpperChar(x)
cmp al, bl
jnz loc_7C9135EE
mov eax, [ebp+arg_4]
jmp short loc_7C92930D
loc_7C91360C:
cmp esi, eax
jnb short loc_7C91361C
mov cl, [esi]
mov dl, [edi]
inc esi
inc edi
cmp cl, dl
jz short loc_7C91360C
jmp short loc_7C9135EE
loc_7C9135EE:
xor al, al
loc_7C91361C:
mov al, 1
jmp short loc_7C9135F0
_RtlEqualString@12 endp
loc_7C9135F0:
pop edi
pop esi
pop ebx
pop ebp
retn 0Ch
ynchronized with Hex View-1)