CMSC 313
COMPUTER ORGANIZATION
&
ASSEMBLY LANGUAGE PROGRAMMING

LECTURE 07, SPRING 2013

TOPICS TODAY

- Project 2
- A Bigger Example

A BIGGER EXAMPLE

Project 1: Escape Sequences

Due: Tuesday October 9, 2001 <--- !!!!!! OLD PROJECT !!!!!

Objective

The objectives of the programming assignment are 1) to gain experience writing larger assembly language programs, and 2) to gain familiarity with various branching operations.

Background

String constants in UNIX and in C/C++ are allowed to contain control characters and other hard-to-type characters. The most familiar of these is '`\n`' for a newline or linefeed character (ASCII code 10). The '`\n`' is called an escape sequence. For this project, we will consider the following escape sequences:

Sequence	**Name**	**ASCII code**
`\a`	alert(bell)	07
`\b`	backspace	08
`\t`	horizontal tab	09
`\n`	newline	10
`\v`	vertical tab	11
`\f`	formfeed	12
`\r`	carriage return	13
`\\`	backslash	92

In addition, strings can have octal escape sequences. An octal escape sequence is a '`\`' followed by one, two or three octal digits. For example, '`\a`' is equivalent to '`\7`' and '`\\`' is equivalent to '`\134`'. Note that in this scheme, the null character can be represented as '`\0`'. The octal escape sequence ends at the third octal digit, before the end of the string, or before the first non-octal digit, whichever comes first. For example "`abc\439xyz`" is equivalent to "`abc#9xyz`" because the ASCII code for '`#`' is 43_8 and 9 is not an octal digit.

Assignment

For this project, you will write a program in assembly language which takes a string input by the user, convert the escape sequences in the string as described above and print out the converted string. In addition, your program should be robust enough to handle user input that might include malformed escape sequences. Examples of malformed escape sequences include: a '`\`' followed by an invalid character, a '`\`' as the last character of the string and a '`\`' followed an octal number that exceeds 255_{10}.

All the invalid escape sequences should be reported to the user (i.e., your program should not just quit after detecting the first invalid escape sequence). When the user input has malformed escape sequences, your program should still convert and print out the rest of the string (which might contain some valid escape sequences). In this case, a '\' should be printed at the location of malformed escape sequence. For example, if the user types in "abc \A def \43 ghi \411" your program should have output:

```
Error: unknown escape sequence \A
Error: octal value overflow in \411
Original: abc \A def \43 ghi \411
Convert:  abc \ def # ghi \
```

Turning in your program

Before you submit your program, record some sample runs of your program using the UNIX script command. You should select sample runs that demonstrate the features supported by your program. Picking good test cases is **your responsibility**.

Use the UNIX '`submit`' command on the GL system to turn in your project. You should submit two files: 1) your assembly language program and 2) the typescript file of your sample runs. The class name for submit is '`cs313`' and the project name is '`proj1`'.

Implementation Issues:

1. You should think carefully about how you will keep track of the number of characters you have already processed in the source string. Since you will process more than one character per iteration of the main loop, you will need a consistent way to update the character count and the pointer into the source string.
2. Your program will have numerous branches. You should think about the layout of your program and how to make it more readable. Avoid *spaghetti code*. Related parts of your program should be placed near each other.
3. Do take into account the fact that the output string might be shorter than the input string.

Notes:

Recall that the project policy states that programming projects must be the result of individual effort. *You are not allowed to work together.* Also, your projects will be graded on five criteria: correctness, design, style, documentation and efficiency. So, it is not sufficient to turn in programs that assemble and run. Assembly language programming can be a messy affair --- neatness counts.

NEXT TIME

- **Stack Instructions**
- **Subroutines**