# Line by Line Parsing in C

Computer Science Department
University of Central Florida

*COP 3502 Recitation Session*

# Parsing

- Typical parsing in C:
  - We read input from keyboard and files as individual tokens separated by white space
    - scanf and fscanf are used for this
      - They read successive tokens from the input
      - They read until white space is encountered and then it stops
      - The next call of scanf picks up from there and reads the next token
  - When is this parsing method useful?
    - If we know how many tokens will be inputed,
    - and we know what each token represents
      - integer, float, string, etc.

## Parsing

- Typical parsing in C:
  - But what if we don’t know how many tokens we will read in?
    - Say it is a list of Math classes for Spring 2010
      - MA 245 MA 318 MA412 …
    - Maybe the list has 10 classes, or 20 classes, or more
  - How would we go about reading this in?

# Parsing

- Typical input files:
  - In these types of files, spaces are usually part of the input
    - Such as the space between a first and last name
  - Tabs and newlines are usually the delimiters
    - Stuff that separates the data

  - Standard processing is to read one entire line at a time
    - Which could have several pieces of information
    - Then use a “string tokenizer” to parse out the different pieces of data in the line.

## Parsing

- How do we make this happen:
  - Start with fgets function:
    - Allows us to read in an entire line at once
      - Meaning, until the next newline
    - char *fgets(char *restrict *s*, int *n*, FILE *restrict *stream*);
      - The first parameter represents the string into which you want to read in the line from the file.
      - The second parameter represents the maximum number of characters you want to read in. (If the line is longer, n characters are read, if the line is shorter, then the whole line is read.)
      - The third parameter is a pointer to the file from which you want to read.
      - The function ALSO returns a pointer to the beginning memory address of the character array into which the line was read.

## Parsing

- How do we make this happen:
  - What do you do with this newly read line:
    - If there is only one item per line, fgets stores that item in the designated character array
      - You then just continue with the program
    - But often files have several pieces of information per line
      - Ex:  Joe Smith, Computer Science, Junior, 3.75
      - So we need to separate out each piece from the newly read line
      - But how?
    - Use a string tokenizer function…

## Parsing

- strtok:
  - In C, the string tokenizer function is strtok:
    - This is a built-in function that we can call
  - The 1st call sets up the string tokenizer
    - You tell the function which string to tokenize,
    - and which items work as delimiters (comma, tab, etc)
  - Example:
    - We read line into an array called line and the delimiters are commas
    - Here’s how you would call the function:
      - strtok(line, “,”);
      - At the end of this call, “line” will just store a string that represents the first token of the original contents

# Parsing

- strtok:
  - To access the remaining tokens:
    - Call the strtok function again, BUT now with a new first parameter
      - Call strtok with NULL as the first parameter and use the same delimiters as in the original call
    - Also, this time, the function will returns a pointer to the beginning of the desired token (the next token)
      - So we must store this pointer.

    - Ex:

```
char *p;
p = strtok(NULL, “,”);
```

## Parsing

- strtok:
  - To access the remaining tokens:
    - You continue making these strtok function calls until there are no more tokens in the line
    - Either you know the number of tokens in the line and simply use a for loop
    - Or, you can check each time to see if the pointer p is NULL or not.
      - If p is NULL, then the function did not return a pointer, meaning there were no more tokens in the string tokenizer

## Parsing

- Additional Information:
  - The function strtok returns a VOID pointer
  - And…your point is…
  - The point is that this pointer needs to be cast to a char pointer
    - More accurate example:

```
char *p;
p = (char*)strtok(NULL, “,”);
```

## Parsing

- Example:

```
#include <stdio.h>
#include <string.h>

int main(void) {
        FILE *fp;  // file pointer
        char line[80];
        char *token;
        char *delimiters = " ,\t\n";  // our delimiters
        char *fn = "data.txt";  // file name
        fp = fopen(fn,"r");

        if (!fp) {
                printf("error opening \"%s\" for reading\n",fn);
                return -1;
        }

        fgets(line, 80, fp);  // grabs the first line

        while (!feof(fp)) {  // checks to make sure the line is not the end of file
                printf("next line\n");
                token = (char*)strtok(line, delimiters); // 1st call
                while (token != NULL) {
                        printf("\tnext token = %s\n",token);
                        token = (char*)strtok(NULL, delimiters); // repeated call
                }
                fgets(line, 80, fp);  // grabs additional lines
        }
        fclose(fp);
        return 0;
}
```

# Parsing

- Example:

<u>So if this was your input:</u>

```
asdf qwer 12345
xyz p q r
() [] !!!
```

<u>Your output would be:</u>

```
next line
        next token = asdf
        next token = qwer
        next token = 12345
next line
        next token = xyz
        next token = p
        next token = q
        next token = r
next line
        next token = ()
        next token = []
        next token = !!!
```

## Parsing

- Other little tidbits:
  - The strtok() function modifies the contents of the original string buffer.
    - Meaning, you will not have access to the original string once you start tokenizing it.
    - So if you need to keep an original copy of the string, you must make this copy yourself using strcpy().

## Parsing

- Other little tidbits:
  - When you use scanf, you do two things:
    - You read in the data till the next white space,
    - AND the data is then parsed accordingly
      - Saved as an int if you used %d, for example
  - Similarly, when you tokenize, you must parse the data properly.
    - atoi() and atof() are two C functions defined in the standard library for this purpose
      - atoi -> ascii-to-int
      - atof -> ascii-to-float

## Parsing

- Other little tidbits:

  Example:

  ```
  char *s = “123”;
  int x = atoi(s);
  ```

  Example:

  ```
  char *t = “3.14159”;
  double y = atof(t);
  ```

  *Note that in spite of its name atof() returns a double value.

# Line by Line Parsing in C

Computer Science Department
University of Central Florida

*COP 3502 Recitation Session*