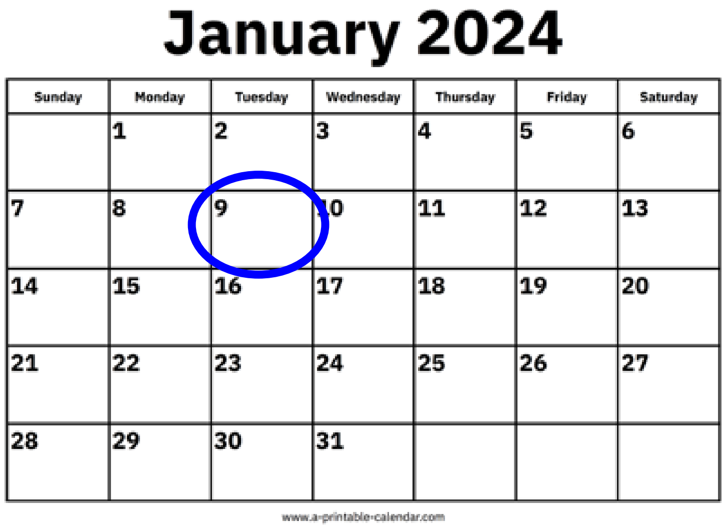
CS 211 : Tues 01/09 (lecture 02)

- Topics: user-defined functions, strings, pointers, arrays

Prof. Hummel
(he/him)



January 2024

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

www.a-printable-calendar.com

Northwestern University

Notes:

- *Lecture slides available on Canvas*
- *Join **replit** if not already (see join link on Canvas)*
- ***HW 01** due tonight @ 11:59pm*
- ***Project 01** due Friday @ 11:59pm, may be submitted up to 48 hours late (see syllabus); Gradescope is open for submissions (4 per day)*

Pointers and strings

- An array is a pointer to the first of N consecutive elements
- A string = array of characters ending with '\0'
- A pointer is a memory address

```c
int main()
{
  int   A[6]; // array of int
  char* s;    // pointer to char
  char  c;
  int*  p;    // pointer to int

  s = "an apple";
  c = s[0];

  p    = A;
  *p   = 10;  // deref ptr
  A[1] = 20;  // deref ptr
  p[2] = 30;  // deref ptr
```

Question #1

1) What does the following program output?

To answer, use iClicker app

(install iClicker Student app, select NU, search for "CS 211")

```
int main()
{
  char* s;
  char  c;

  s = "ACE";
  c = s[0];
  c = c + 1;

  printf("%d\n", c);

  // printf("%c\n", c);
  // printf("%b\n", c);
```

A) ACF

B) ACE1

C) BCE

D) B

E) 66

Question #2

2) Which one correctly outputs the # of characters in the string "apple"?

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

```c
int main()
{
  char* s;
  s = "apple";


  printf("%lu\n", sizeof("apple"));  // A
  printf("%lu\n", sizeof(s));        // B
  printf("%lu\n", strlen(s));        // C
```

A) sizeof("apple")

B) sizeof(s)

C) strlen(s)

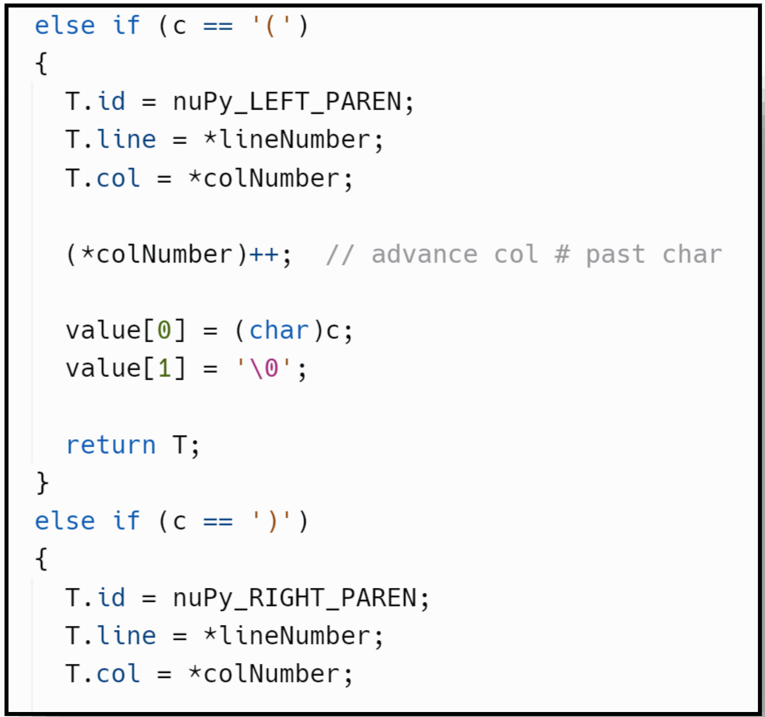
D) None of these

Programming is Hard

- **Working like an engineer can help…**
  - *Build in* ***steps****, not all at once*
  - *Reduces frustration*
  - *Builds confidence*

Project 01

- Pick a token to recognize
- Add if statement to the chain…
- Run and test interactively…



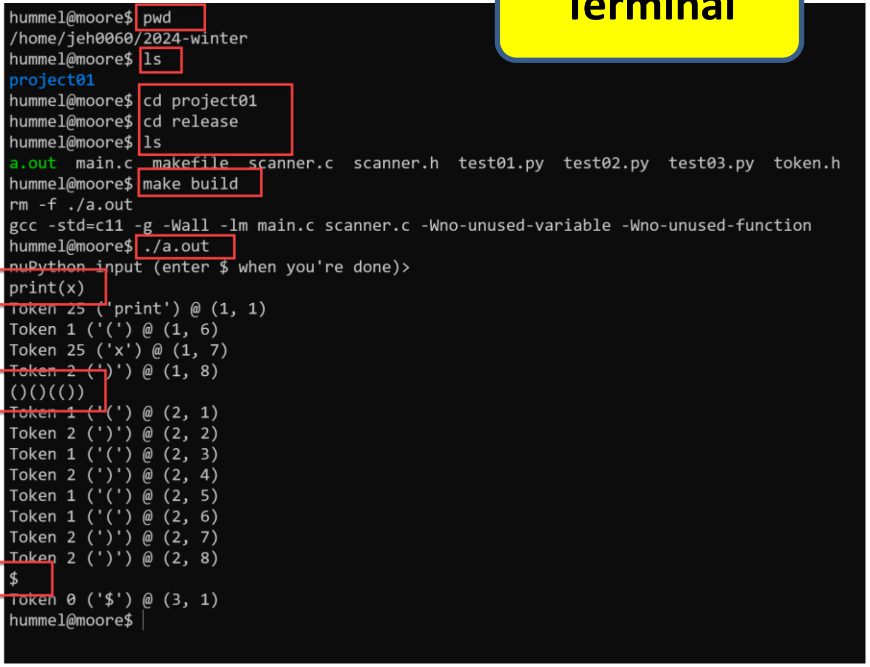
```
else if (c == '(')
{
  T.id = nuPy_LEFT_PAREN;
  T.line = *lineNumber;
  T.col = *colNumber;

  (*colNumber)++;  // advance col # past char

  value[0] = (char)c;
  value[1] = '\0';

  return T;
}
else if (c == ')')
{
  T.id = nuPy_RIGHT_PAREN;
  T.line = *lineNumber;
  T.col = *colNumber;
```

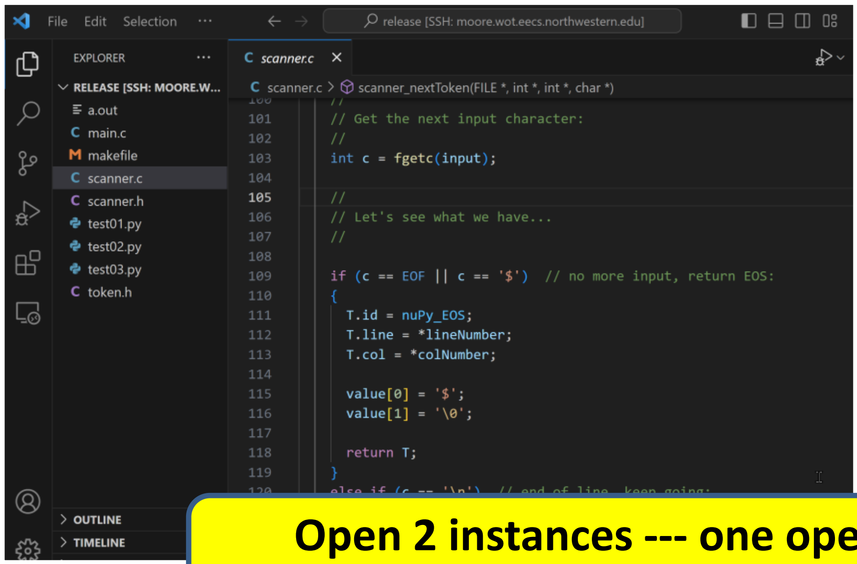
Terminal



```
hummel@moore$ pwd
/home/jeh0060/2024-winter
hummel@moore$ ls
project01
hummel@moore$ cd project01
hummel@moore$ cd release
hummel@moore$ ls
a.out  main.c  makefile  scanner.c  scanner.h  test01.py  test02.py  test03.py  token.h
hummel@moore$ make build
rm -f ./a.out
gcc -std=c11 -g -Wall -lm main.c scanner.c -Wno-unused-variable -Wno-unused-function
hummel@moore$ ./a.out
nuPython input (enter $ when you're done)>
print(x)
Token 25 ('print') @ (1, 1)
Token 1 ('(') @ (1, 6)
Token 25 ('x') @ (1, 7)
Token 2 (')') @ (1, 8)
()()(())
Token 1 ('(') @ (2, 1)
Token 2 (')') @ (2, 2)
Token 1 ('(') @ (2, 3)
Token 2 (')') @ (2, 4)
Token 1 ('(') @ (2, 5)
Token 1 ('(') @ (2, 6)
Token 2 (')') @ (2, 7)
Token 2 (')') @ (2, 8)
$
Token 0 ('$') @ (3, 1)
hummel@moore$
```

Why VS Code + EECS computers?

- Visual Studio Code is a flexible & popular editor
- EECS computers provide C/C++ software we need
- Learn command-line / Linux
- Prep for CS 213 / CS 343
- Avoid C / C++ installation issues



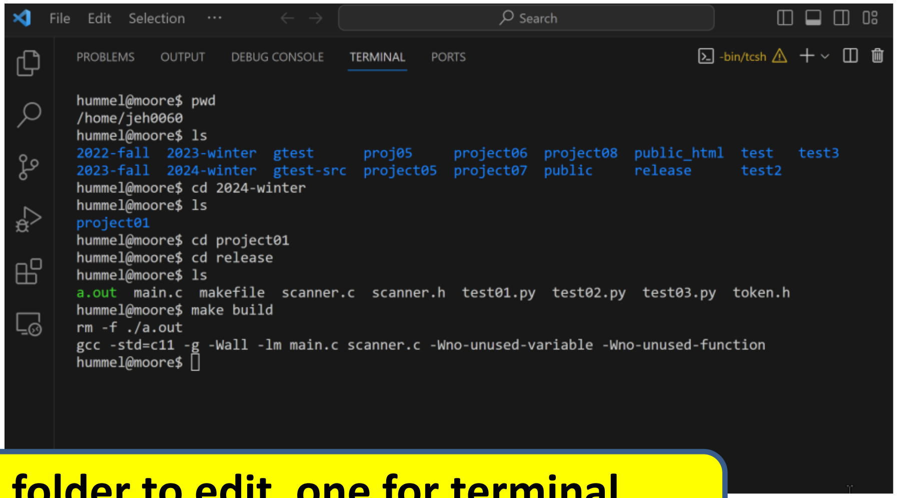
```
    //
    // Get the next input character:
    //
    int c = fgetc(input);

    //
    // Let's see what we have...
    //

    if (c == EOF || c == '$')  // no more input, return EOS:
    {
      T.id = nuPy_EOS;
      T.line = *lineNumber;
      T.col = *colNumber;

      value[0] = '$';
      value[1] = '\0';

      return T;
    }
    else if (c == '\n')  // end of line, keep going:
```



```
hummel@moore$ pwd
/home/jeh0060
hummel@moore$ ls
2022-fall  2023-winter  gtest      proj05     project06  project08  public_html  test   test3
2023-fall  2024-winter  gtest-src  project05  project07  public     release      test2
hummel@moore$ cd 2024-winter
hummel@moore$ ls
project01
hummel@moore$ cd project01
hummel@moore$ cd release
hummel@moore$ ls
a.out  main.c  makefile  scanner.c  scanner.h  test01.py  test02.py  test03.py  token.h
hummel@moore$ make build
rm -f ./a.out
gcc -std=c11 -g -Wall -lm main.c scanner.c -Wno-unused-variable -Wno-unused-function
hummel@moore$ 
```

Open 2 instances --- one opens folder to edit, one for terminal
(if moore is slow, try batgirl or batman@eecs.northwestern.edu)

Functions help manage complexity

- Functions are building blocks
- We've seen a number of functions already:

`pow(W, 0.16)`

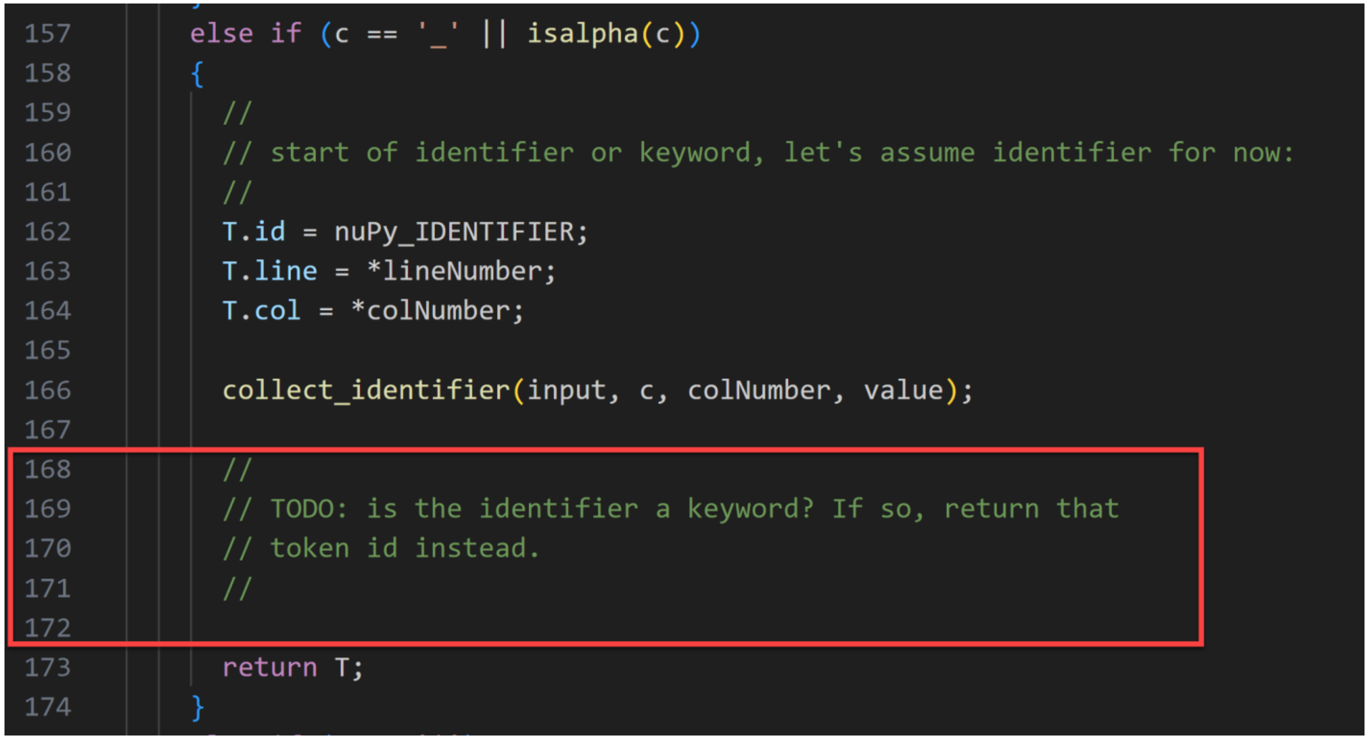
`strlen(s)`

`printf("…")`

`scanf("%d", &x);`

Example from Project 01

- **Determining if an identifier is a keyword…**
  - *"x" is an identifier*
  - *"print" is an identifier*
  - *"if" is a keyword*



```
else if (c == '_' || isalpha(c))
{
  //
  // start of identifier or keyword, let's assume identifier for now:
  //
  T.id = nuPy_IDENTIFIER;
  T.line = *lineNumber;
  T.col = *colNumber;

  collect_identifier(input, c, colNumber, value);

  //
  // TODO: is the identifier a keyword? If so, return that
  // token id instead.
  //

  return T;
}
```

id_or_keyword( ) function

Open VS Code or replit for project 01 and type along. Feel free to use this code…

```
//
// id_or_keyword
//
// Given a value like "x" or "if", returns whether this value is a
// nuPython keyword or a nuPython identifier. Returns the appropriate
// Token id: nuPy_IDENTIFIER, nuPy_KEYW_AND, nuPy_KEYW_BREAK, etc.
//
static int id_or_keyword(char* value)
{
  assert(strlen(value) > 0);  // valid value?

  //
  // NOTE: array elements must be in the SAME ORDER as the
  // keywords in the tokens.h enum.
  //
  char* keywords[] = {"and", "break", "continue", "def", "elif", "else",
                      "False", "for", "if", "in", "is", "None", "not",
                      "or", "pass", "return", "True", "while"};
  .
  .
  .
}
```

Solution

```
//
// id_or_keyword
//
// Given a value like "x" or "if", returns whether this value is a
// nuPython keyword or a nuPython identifier. Returns the appropriate
// Token id: nuPy_IDENTIFIER, nuPy_KEYW_AND, nuPy_KEYW_BREAK, etc.
//
static int id_or_keyword(char* value)
{
  assert(strlen(value) > 0);  // valid value?

  // NOTE: array elements must be in the SAME ORDER as the
  // keywords in the tokens.h enum.
  char* keywords[] = {"and", "break", "continue", "def", "elif", "else",
                      "False", "for", "if", "in", "is", "None", "not",
                      "or", "pass", "return", "True", "while"};

  int N = sizeof(keywords) / sizeof(keywords[0]);

  int index = -1; // index where we find it, assume not found initially

  for (int i = 0; i < N; i++) {
    if (strcmp(value, keywords[i]) == 0) { // match!
      index = i;
      break;
    }
  }

  if (index < 0)
    return nuPy_IDENTIFIER;
  else
    return nuPy_KEYW_AND + index;
}
```

Observations...

```c
static int id_or_keyword(char* value)
{
  assert(strlen(value) > 0);  // valid value?

  // NOTE: array elements must be in the SAME ORDER as the
  // keywords in the tokens.h enum.
  char* keywords[] = {"and", "break", "continue", "def", "elif", "else",
                      "False", "for", "if", "in", "is", "None", "not",
                      "or", "pass", "return", "True", "while"};

  int N = sizeof(keywords) / sizeof(keywords[0]);

  int index = -1; // index where we find it, assume not found initially

  for (int i = 0; i < N; i++) {
    if (strcmp(value, keywords[i]) == 0) { // match!
      index = i;
      break;
    }
  }

  if (index < 0)
    return nuPy_IDENTIFIER;
  else
    return nuPy_KEYW_AND + index;
}
```

1. **Passed a string parameter, returns an int**
2. **"static" => local helper function**
3. **"assert" => defensive programming**
4. **Linear search**
5. **For loops are great for counting --- e.g. array indices**

Visualization exercise

- Draw what you think memory looks like for the **keywords** data structure…

```
//
// NOTE: array elements must be in the SAME ORDER as the
// keywords in the tokens.h enum.
//
char* keywords[] = { "and", "break", "continue", ..., "while" };
```

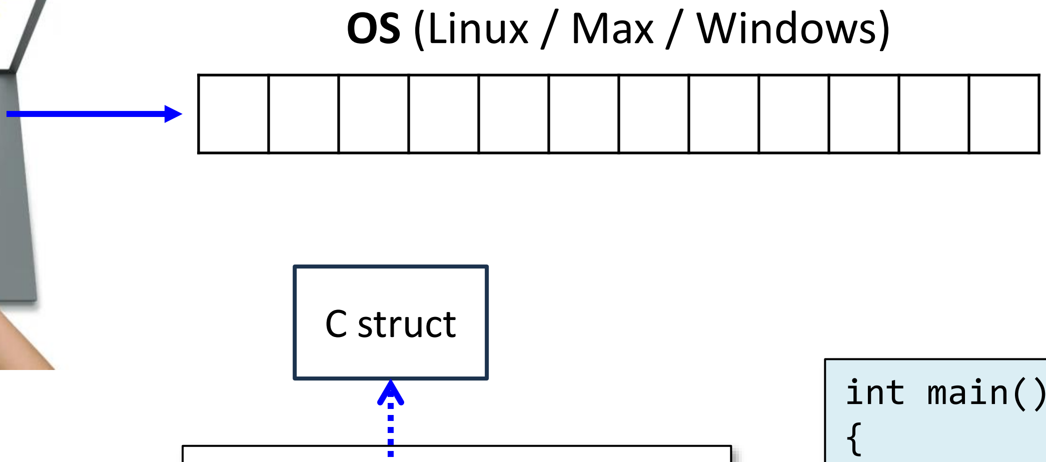
Tool: https://sketch.io/sketchpad/

Discussion

```
//
// NOTE: array elements must be in the SAME ORDER as the
// keywords in the tokens.h enum.
//
char* keywords[] = { "and", "break", "continue", ..., "while" };
```

How does input work?

- **Input is buffered by the operating system…**
- **C provides functions that advance through buffer…**



```
FILE* stdin; // keyboard

int fgetc(FILE* f)
{
   .
   .
   .
}
```

Each call to **fgetc( )** advances buffer pointer…

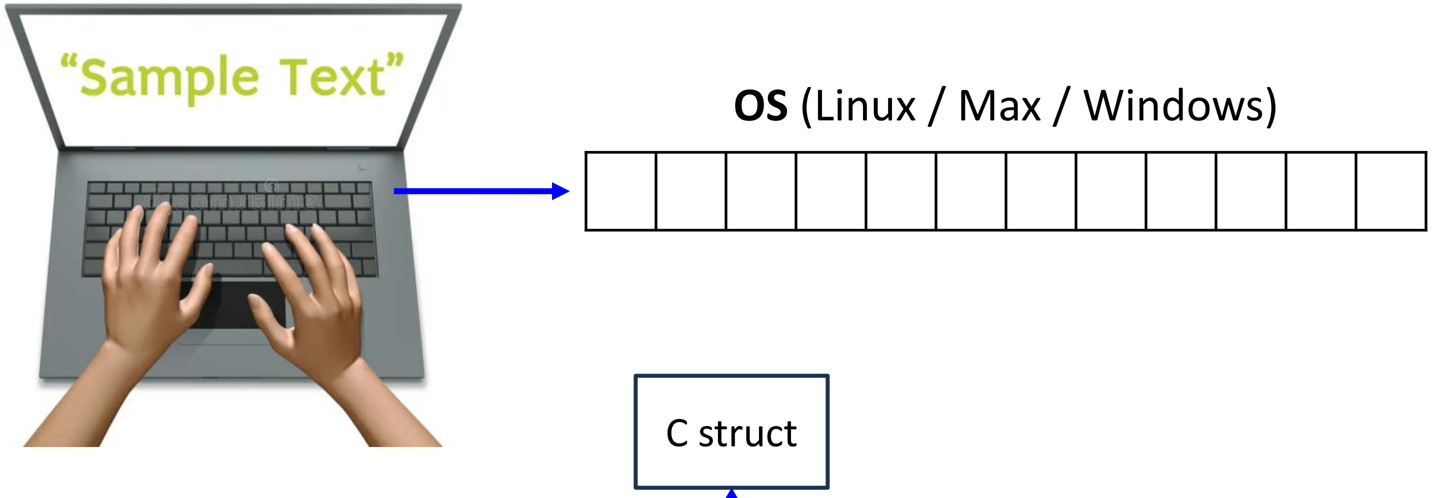
```
int main()
{
  printf("Enter a line:\n");

  int c = fgetc(stdin);

  while (c != '\n')
  {
     printf("'%c'\n", c);
     c = fgetc(stdin);
  }
```

ungetc( )

- **ungetc(c, stdin) puts c back into the keyboard buffer…**



```
FILE* stdin; // keyboard

int ungetc(int c, FILE* f)
{
   .
   .
   .
}
```

```
int main()
{
   int c;
   c = fgetc(stdin);
   c = fgetc(stdin);

   ungetc('$', stdin);
```

Side-effects

- **fgetc( ) and ungetc( ) are examples of functions with "side-effects"**
- **Calling these functions cause internal state changes**
  - *The buffer and buffer pointer potentially change…*

Project 01: string literals

- **Here's an approach for handling string literals "..."**

```
if (c == '"')  // start of a string literal "..."
{
   T.id   = …
   T.line = …
   T.col  = …

   int i = 0;

   while (fgetc(input) != '"' && fgetc(input) != '\n') {

      // not yet at end, store and repeat:
      value[i] = fgetc(input);
      i++;

      (*colNumber)++;
   }
```

Question #3

3) Suppose the user types "apple" without the quotes and presses ENTER. What is output?

```
int main()
{
  int c;

  while (fgetc(stdin) != 'e') {

     c = fgetc(stdin);
     printf("%c,", c);
  }

  return 0;
}
```

A) a,p,p,l,e,

B) a,p,p,l,

C) a,p,e

D) a,p,

E) p,l,

What should I be working on?

1. ***HW #01*** *is due tonight @ 11:59pm…*
2. ***Project #01*** *is due Friday @ 11:59pm…*