APS 105 Lecture Notes 23

Last lecture: Introduce Strings

Today: How to pass strings to functions and how to printf a string, and how to take string from the user

Recap:

if you have

```
char *p = "Wow";
```

CAN DO `p = "Other";` // will make p point to another const. string array

CANNOT DO `p[0] = 'Q';` // since "Wow" is constant, so can't change individual elements.
→ Run-time error ←

```
char s[] = "Fine";
p = s;
```

CAN DO `p[0] = 's';` `s[1] = 'q';` // since "Fine" is not constant, it is a string in the "stack"

CANNOT DO `s = "Hello";` // s is an address/array identifier not pointer value, so you can't change it.
→ Compile-time error ←

Write a function that counts # of spaces in a string

```
int spaceCounter(char *s){        // or char s[]
    int count = 0;
    for (int i=0; s[i]!='\0'; i++){
        if (s[i] == ' ') count++;
    return count;
}
```

This is more common

What is the usage of '\0'?

To know the end of a string.

In printing strings, printf will print characters until it observes a null character

```
char *s = "An example";
printf("Sample string is %s", s);
```

format specifier of string.

Will print

Sample string is An example

```
printf("Sample string is %.4s", s);
```

prints 4 characters

prints Sample string is An␣e

1 2 3 4

```
char *p = "Sample";
printf("%s \n", p);
printf("%s \n", p+2);
```

Sample
will print starting p[0] till it sees a '\0'
mple
will print starting p[2] till it sees a '\0'

```
printf("%c", *p);
```
S

```
printf("%c", *(p+2));
```
m

We also have puts(s); → 1 argument

Input string:

Using scanf

```
char st[10];
scanf("%s", st);   //why not &st
```

The prototype of scanf("<format specifier>", address of variable);

st → points to 1st element in array.

* scanf will ignore white spaces before characters
* scanf will read until a white space (spaces, tab, return/enter)
* scanf will terminate the st array with '\0' good news

→ Hence typing ␣␣␣ABCD <enter>

will save ABCD in st

→ If you type ␣␣␣ABCD␣␣␣f f <enter>

st will store ABCD, and ff will be read by the next scanf

gets (s); //just like scanf, but gets reads leading white spaces until an <enter>, it excludes '\n' in character

Problem: (A major one)

If char st[7+1];

scanf("%s", st);

And we type ABCDEFGHI

We typed 9 characters to store in 8 character array, the characters will overflow the storage allocated in a phenomenon called "buffer overflow". So extra characters will be stored on stack after the last element in st.

↓ Unsafe as it can change values of other variables

What is safe?

fgets (string, # of chars, stdin);

- string: identifier
- # of chars: clearly say how many characters to read: # of chars − 1 to leave 1 for '\0'
- stdin: standard file where input is being saved.

* There is a function that only reads 1 character

getchar() and returns the character read.

```
char c;
c = getchar();
```

equivalent ⟺ scanf("%c", c);

Write a function that safely reads from the user

↓

meaning reads only size-1 characters and 1 character is left for '\0'

```
int main(void){
    char s[6];
    printf("User is entering .... %s", getStringSafely(s,6));
    return 0;
}
char * getStringSafely(char *s, int size){
    int i=0;
    char c;
    while (i<size-1 && (c=getchar()) != '\n'){
        s[i]= c;
        i=i+1;
    }
    s[i]= '\0';
    return s;
}
```