

# CS 33

## Introduction to C Part 5

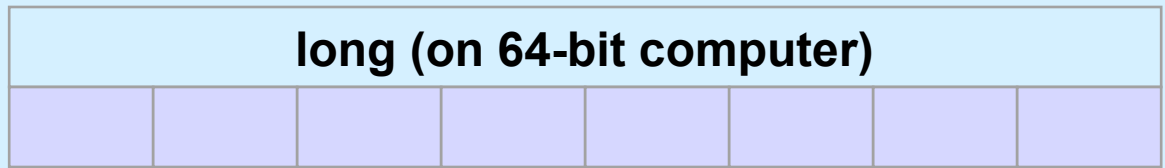
# Basic Data Types



$-2,147,483,648 - 2,147,483,647$



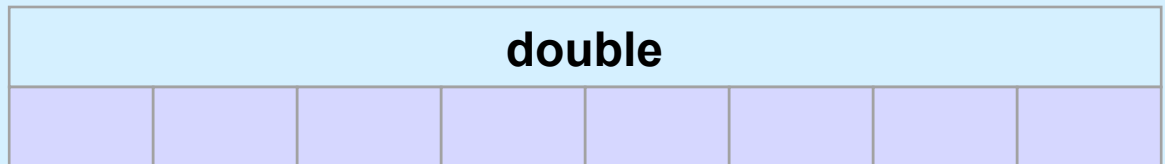
$-32,768 - 32,767$



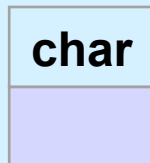
$-9,223,372,036,854,775,808 - 9,223,372,036,854,775,807$



$\sim 10^{-44.85} - \sim 10^{38.53}$ , 23-bit significand



$\sim 10^{-323.3} - \sim 10^{308.3}$ , 52-bit significand



$-128 - 127$

# Characters

- **ASCII**

- **American Standard Code for Information Interchange**

- **works for:**

- » **English**
    - » **Swahili**

- » **not much else**

- **doesn't work for:**

- » **French**
    - » **Dutch**
    - » **Spanish**
    - » **German**

- » **Arabic**
    - » **Sanskrit**
    - » **Chinese**
    - » **pretty much everything else**



# Who cares!!

---



**You should care ...  
(but not in this course)**

# ASCII Character Set

|    | 00    | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |
|----|-------|----|----|----|----|----|----|----|----|----|-----|-----|-----|
|    | ----- |    |    |    |    |    |    |    |    |    |     |     |     |
| 0: | \0    | \n |    | (  | 2  | <  | F  | P  | Z  | d  | n   | x   |     |
| 1: |       | \v |    | )  | 3  | =  | G  | Q  | [  | e  | o   | y   |     |
| 2: |       | \f | sp | *  | 4  | >  | H  | R  | \  | f  | p   | z   |     |
| 3: |       | \r | !  | +  | 5  | ?  | I  | S  | ]  | g  | q   | {   |     |
| 4: |       |    | "  | ,  | 6  | @  | J  | T  | ^  | h  | r   |     |     |
| 5: |       |    | #  | -  | 7  | A  | K  | U  | _  | i  | s   | }   |     |
| 6: |       |    | \$ | .  | 8  | B  | L  | V  | `  | j  | t   | ~   |     |
| 7: | \a    |    | %  | /  | 9  | C  | M  | W  | a  | k  | u   | DEL |     |
| 8: | \b    |    | &  | 0  | :  | D  | N  | X  | b  | l  | v   |     |     |
| 9: | \t    |    | '  | 1  | ;  | E  | O  | Y  | c  | m  | w   |     |     |

# *chars* as Integers

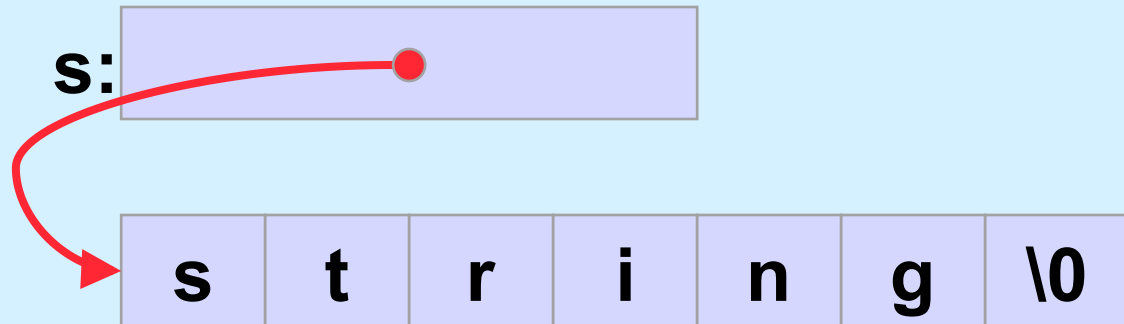
```
char tolower(char c) {  
    if (c >= 'A' && c <= 'Z')  
        return c + 'a' - 'A';  
    else  
        return c;  
}
```

# Character Strings

```
char c = 'a';
```

**c:** a

```
char *s = "string";
```






**Is there any difference between *c1* and *c2* in the following?**

```
char c1 = 'a';
```

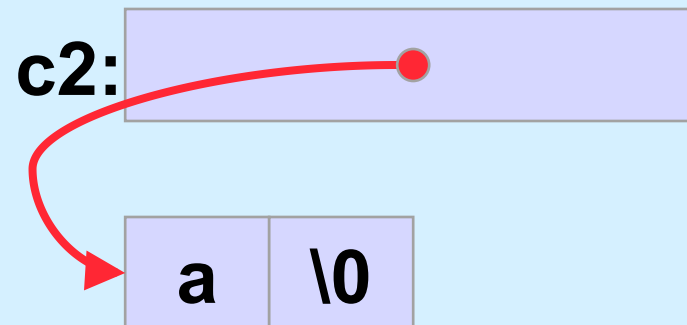
```
char *c2 = "a";
```

# Yes!!

```
char c1 = 'a';
```

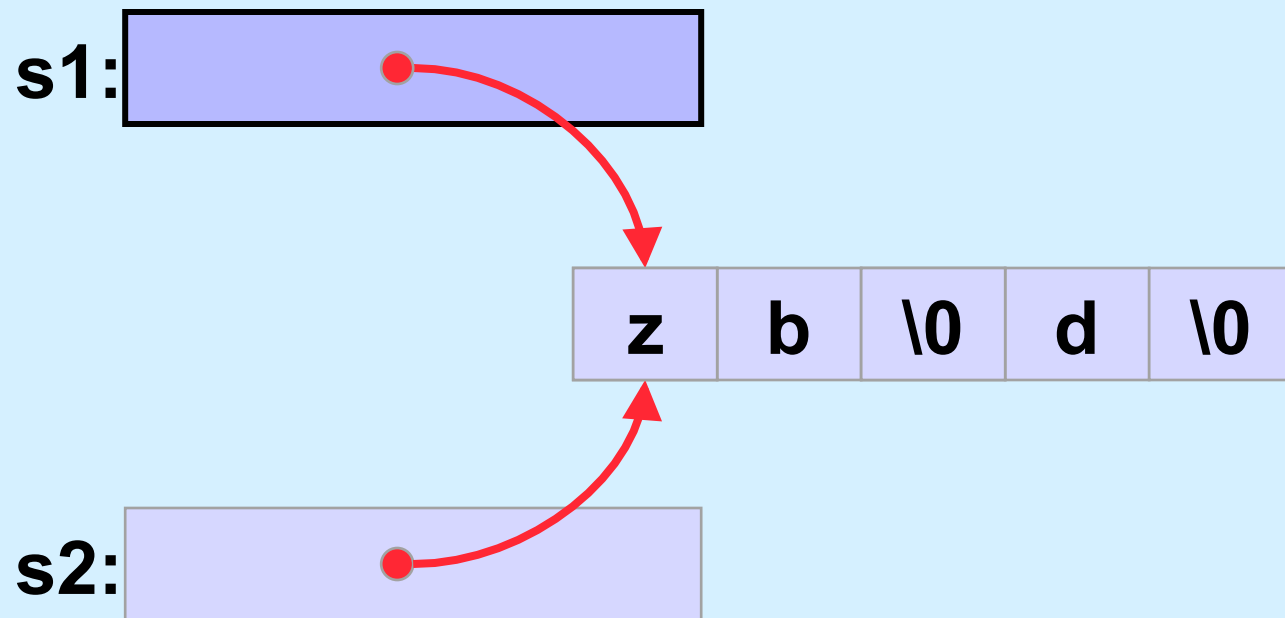
**c1:** 

```
char *c2 = "a";
```



**What do *s1* and *s2* refer to after the following is executed?**

```
char s1[] = "abcd";  
char *s2 = s1;  
s1[0] = 'z';  
s2[2] = '\\0';
```



# Weird ...

Suppose we did it this way:

```
char *s1 = "abcd";  
char *s2 = s1;  
s1[0] = 'z';  
s1[2] = '\\0';
```

```
% gcc -o char char.c
```

```
% ./char
```

Segmentation fault



# Copying Strings (1)

```
char s1[] = "abcd";
```

```
char s2[5];
```

```
s2 = s1;    // does this do anything useful?
```

```
// correct code for copying a string
```

```
for (i=0; s1[i] != '\0'; i++)
```

```
    s2[i] = s1[i];
```

```
s2[i] = '\0';
```

```
// would it work if s2 were declared:
```

```
char *s2;
```

```
// ?
```

# Copying Strings (2)

```
char s1[] = "abcdefghijklmnopqrstuvwxyz";
```

```
char s2[5];
```

```
for (i=0; s1[i] != '\0'; i++)
```

```
    s2[i] = s1[i];
```

```
s2[i] = '\0';
```

**Does this work?**

```
for (i=0; (i<4) && (s1[i] != '\0'); i++)
```

```
    s2[i] = s1[i];
```

```
s2[i] = '\0';
```

# String Length

```
char *s1;
```

```
s1 = produce_a_string();  
// how long is the string?
```

```
sizeof(s1); // doesn't yield the length!!
```

```
for (i=0; s1[i] != '\0'; i++)  
    ;  
// number of characters in s1 is i
```



# Size

```
int main() {  
    char s[] = "1234";  
    printf("%d\n", sizeof(s));  
    proc(s, 5);  
    return 0;  
}
```

```
$ gcc -o size size.c  
$ ./size  
5  
8  
12  
$
```

```
void proc(char s1[], int len) {  
    char s2[12];  
    printf("%d\n", sizeof(s1));  
    printf("%d\n", sizeof(s2));  
}
```

# Quiz 1

```
void proc(char s[16]) {  
    printf("%d\n", sizeof(s));  
}
```

**What's printed?**

- a) 8
- b) 15
- c) 16
- d) 17

# Comparing Strings (1)

```
char *s1;
```

```
char *s2;
```

```
s1 = produce_a_string();
```

```
s2 = produce_another_string();
```

```
// how can we tell if the strings are the same?
```

```
if (s1 == s2) {
```

```
    // does this mean the strings are the same?
```

```
} else {
```

```
    // does this mean the strings are different?
```

```
}
```

# Comparing Strings (2)

```
int strcmp(char *s1, char *s2) {  
    for (i=0;  
        (s1[i] == s2[i]) && (s1[i] != 0) && (s2[i] != 0);  
        i++)  
        ; // an empty statement  
    if (s1[i] == 0) {  
        if (s2[i] == 0) return 0; // strings are identical  
        else return -1; // s1 < s2  
    } else if (s2[i] == 0) return 1; // s2 < s1  
    if (s1[i] < s2[i]) return -1; // s1 < s2  
    else return 1; // s2 < s1;  
}
```

# The String Library

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

```
char *strcpy(char *dest, char *src);
```

```
    // copy src to dest, returns ptr to dest
```

```
char *strncpy(char *dest, char *src, int n);
```

```
    // copy at most n bytes from src to dest
```

```
int strlen(char *s);
```

```
    // return the length of s (not counting the null)
```

```
int strcmp(char *s1, char *s2);
```

```
    // returns -1, 0, or 1 depending on whether s1 is
```

```
    // less than, the same as, or greater than s2
```

```
int strncmp(char *s1, char *s2, int n);
```

```
    // do the same, but for at most n bytes
```

# The String Library (more)

```
size_t strspn(const char *s, const char *accept);  
    // returns length of initial portion of s  
    // consisting entirely of bytes from accept
```

```
size_t strcspn(const char *s, const char *reject);  
    // returns length of initial portion of s  
    // consisting entirely of bytes not from  
    // reject
```

# Quiz 2

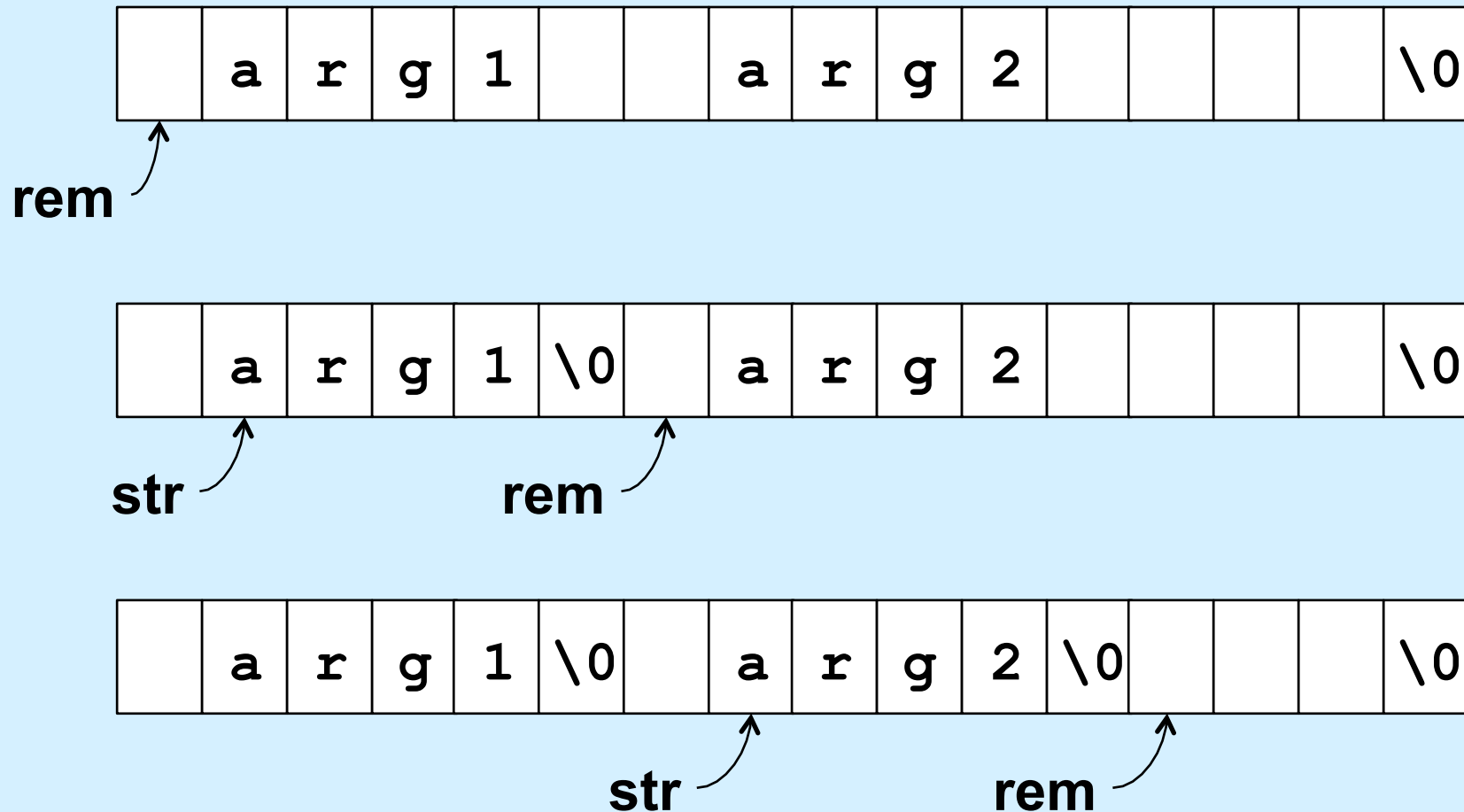
```
#include <stdio.h>
#include <string.h>
```

```
int main() {
    char s1[] = "Hello World!\n";
    char *s2;
    strcpy(s2, s1);
    printf("%s", s2);
    return 0;
}
```

**This code:**

- a) is a great example of well written C code**
- b) has syntax problems**
- c) might seg fault**

# Parsing a String





# Design of *getfirstword*

- **char \*getfirstword(char \*\*strp)**
  - returns
    - » pointer to null-terminated first word in \*strp
    - or
    - » NULL, if \*strp points to a string of entirely whitespace
  - \*strp modified to
    - » point to character following first word in \*strp if within bounds of string
    - or
    - » NULL if next character not within bounds

# Code

```
char *getfirstword(char **strp) {
    char *str = *strp;
    if (str == NULL)
        return NULL;
    int len = strlen(str);
    int wslen =
        strspn(str, " \t\n");
        // initial whitespace
    if (wslen == len) {
        // string is all whitespace
        return NULL;
    }
    str = &str[wslen];
    // skip over whitespace
    len -= wslen;

    int wlen =
        strcspn(str, " \t\n");
        // length of first word
    if (wlen < len) {
        // word ends before end of
        // string: terminate
        // it with null
        str[wlen] = '\0';
        *strp = &str[wlen+1];
    } else {
        // no more words
        *strp = NULL;
    }
    return str;
}
```