CS 33

Introduction to C
Part 5
Structures vs. Objects

- Are structs objects?

**NO!**

(What’s an object?)
# Structures Containing Arrays

```c
struct Array {
    int A[6];
} S1, S2;

int A1[6], A2[6];

A1 = A2;
    // not legal: arrays don’t know how big they are

S1 = S2;
    // legal: structures do
```
A Bit More Syntax …

- Constants

```c
const double pi = 3.141592653589793238;

area = pi*r*r;    /* legal */
pi = 3.0;         /* illegal */
```
More Syntax ...

```c
const int six = 6;
int nonconstant;
const int *ptr_to_constant;
int *const constant_ptr = &nonconstant;
const int *const constant_ptr_to_constant = &six;

ptr_to_constant = &six;
    // ok
*ptr_to_constant = 7;
    // not ok
*constant_ptr = 7;
    // ok
constant_ptr = &six;
    // not ok
```
And Still More ...

- Array initialization

```c
int SomeMorePrimes[] = {17, 19, 23, 29};
int MoreWithRoomForGrowth[10] = {31, 37};
int MagicSquare[][] = {{2, 7, 6},
                      {9, 5, 1},
                      {4, 3, 8}};
```
Basic Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
<td>-2,147,483,648 – 2,147,483,647</td>
</tr>
<tr>
<td>short</td>
<td>-32,768</td>
<td>32,767</td>
<td>-32,768 – 32,767</td>
</tr>
<tr>
<td>char</td>
<td>-128</td>
<td>127</td>
<td>-128 – 127</td>
</tr>
<tr>
<td>float</td>
<td>±1.8 × 10⁻³⁸</td>
<td>±3.4 × 10³⁸</td>
<td>±1.8×10⁻³⁸ – ±3.4×10³⁸, ~7 decimal digits</td>
</tr>
<tr>
<td>double</td>
<td>±2.23 × 10⁻³⁰⁸</td>
<td>±1.8 × 10³⁸</td>
<td>±2.23×10⁻³⁰⁸ – ±1.8×10³⁸, ~16 decimal digits</td>
</tr>
</tbody>
</table>
Character

- **ASCII**
  - American Standard Code for Information Interchange
  - works for:
    - English
    - Swahili
  - doesn’t work for:
    - French
    - Spanish
    - German
    - Korean
    - Arabic
    - Sanskrit
    - Chinese
    - pretty much everything else
Characters

• Unicode
  – support for the rest of world
  – defines a number of encodings
  – most common is UTF-8
    » variable-length characters
    » ASCII is a subset and represented in one byte
    » larger character sets require an additional one to three bytes
  – not covered in CS 33
## ASCII Character Set

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

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

char c = 'a';

char *s = "string";

s: string string string string string string string string \0
Is there any difference between \texttt{c1} and \texttt{c2} in the following?

\begin{verbatim}
char c1 = 'a';
char *c2 = "a";
\end{verbatim}
Yes!!

```c
char c1 = 'a';

char *c2 = "a";
```

```
c1: a

c2: a \0
```
What do \texttt{s1} and \texttt{s2} refer to after the following is executed?

\begin{verbatim}
char s1[] = "abcd";
char *s2 = s1;
s1[0] = 'z';
s2[2] = '\0';
\end{verbatim}
Weird ...

Suppose we did it this way:

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

% gcc -o char char.c

% ./char

Segmentation fault
Copying Strings (1)

```c
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)

```c
char s1[] = "abcdefghijklmnopqrstuvwxyz";
char s2[5];

for (i=0; s1[i] != '\0'; i++)
    s2[i] = s1[i];
s2[i] = '\0';

for (i=0; (i<4) && (s1[i] != '\0'); i++)
    s2[i] = s1[i];
s2[i] = '\0';
```

Does this work?

Works!
String Length

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

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

```c
int strcmp(char *s1, char *s2) {
    int i;
    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)

```c
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

<table>
<thead>
<tr>
<th>arg1</th>
<th>arg2</th>
<th>\0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arg1</td>
<td>\0</td>
<td>arg2</td>
</tr>
<tr>
<td>str</td>
<td>rem</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arg1</td>
<td>\0</td>
<td>arg2</td>
</tr>
<tr>
<td>str</td>
<td>rem</td>
<td></td>
</tr>
</tbody>
</table>
Design of `getfirstword`

- `char *getfirstword(char **rem_p)`
  - returns
    - pointer to null-terminated first word in `*rem_p`
    - or
      - `NULL`, if `*rem_p` is a string entirely of whitespace
  - `*rem_p` modified to
    - point to character following first word in `*rem_p` if within bounds of string
    - or
      - `NULL` if next character not within bounds
Using *getfirstword*

```c
int main() {
    char line[] = " arg0  arg1 arg2  arg3 ";
    char *rem = line;
    char *str;
    while ((str = getfirstword(&rem)) != NULL) {
        printf("%s
", str);
    }
    return 0;
}
```

*Output:*

```
arg0
arg1
arg2
arg3
```
```c
char *getfirstword(char **rem_p) {
    char *str = *rem_p;
    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';
        *rem_p = &str[wlen+1];
    } else {
        // no more words
        *rem_p = NULL;
    }
    return str;
}
```