CS 33

Introduction to C
Part 5
## Basic Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>-2,147,483,648 – 2,147,483,647</td>
<td>±1.8×10⁻³⁸ – ±3.4×10³⁸, ~7 decimal digits</td>
</tr>
<tr>
<td>short</td>
<td>-32,768 – 32,767</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>-128 – 127</td>
<td></td>
</tr>
<tr>
<td>long</td>
<td>-9,223,372,036,854,775,808 – 9,223,372,036,854,775,807</td>
<td>±2.23×10⁻³⁰⁸ – ±1.8×10³⁰⁸, ~16 decimal digits</td>
</tr>
<tr>
<td>float</td>
<td></td>
<td></td>
</tr>
<tr>
<td>double</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Characters

• 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

| 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

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

```c
char c = 'a';

char *s = "string";
```

```
c: a

s: string
```

Copyright © 2017 Thomas W. Doeppner. All rights reserved.
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';
```

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

Diagram:
- `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
```
# Size

```c
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));
}
```

$ gcc -o size size.c$
$ ./size
5
8
12
$
Quiz 1

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

What’s printed?

a) 8  
b) 15  
c) 16  
d) 17
Comparing Strings (1)

```c
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

```c
#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

```c
#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

```
arg1 arg2 \0
```

```
arg1 \0 arg2 \0
```

```
arg1 \0 arg2 \0 \0
```
Design of \textit{getfirstword}

- \textbf{char \*getfirstword(char **rem\_p)}
  - returns
    - pointer to null-terminated first word in \textit{rem\_p}
    - or
    - NULL, if \textit{rem\_p} is a string entirely of whitespace
  - \textit{rem\_p} modified to
    - point to character following first word in \textit{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\n", str);
    }
    return 0;
}
```

Output:
- `arg0`
- `arg1`
- `arg2`
- `arg3`
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
    str[wlen] = '\0';
    *rem_p = &str[wlen+1];
} else {
    // no more words
    *rem_p = NULL;
}
return str;