CS 33
Introduction to C
Part 5
Digression: Where Stuff Is (Roughly)

- Code (aka text)
- Global and Static Local Data
- Stack

Virtual Memory
scanf: Reading Data

```c
int main() {
    int i, j;
    scanf("%d %d", &i, &j);
    printf("%d, %d", i, j);
}
```

Two parts

• formatting instructions
  – whitespace in format string matches any amount of whitespace in input
    » whitespace is space, tab, newline (‘\n’)

• arguments: must be addresses
  – why?

$ ./a.out
3        12
3, 12
#define (again)

```c
#define CtoF(cel) (9.0*cel)/5.0 + 32.0
```

Simple textual substitution:

```c
float tempc = 20.0;
float tempf = CtoF(tempc);
// same as tempf = (9.0*tempc)/5.0 + 32.0;
```
#define CtoF(cel) ((9.0*cel)/5.0 + 32.0)

float tempc = 20.0;
float tempf = CtoF(tempc+10);
// same as tempf = (9.0*(tempc+10))/5.0 + 32.0;

#define CtoF(cel) ((9.0*(cel))/5.0 + 32.0)

float tempc = 20.0;
float tempf = CtoF(tempc+10);
// same as tempf = (9.0*(tempc+10))/5.0 + 32.0;
Structures

```c
struct ComplexNumber {
    float real;
    float imag;
};

struct ComplexNumber x;
x.real = 1.4;
x.imag = 3.65e-10;
```
Pointers to Structures

```c
struct ComplexNumber {
    float real;
    float imag;
};

struct ComplexNumber x, *y;
x.real = 1.4;
x.imag = 3.65e-10;
y = &x;
y->real = 2.6523;
y->imag = 1.428e20;
```
structs and Functions

```
struct ComplexNumber ComplexAdd(
    struct ComplexNumber a1,
    struct ComplexNumber a2) {
    struct ComplexNumber result;
    result.real = a1.real + a2.real;
    result.imag = a1.imag + a2.imag;
    return result;
}
```
Would This Work?

```c
struct ComplexNumber *ComplexAdd(
    struct ComplexNumber *a1,
    struct ComplexNumber *a2) {
    struct ComplexNumber result;
    result.real = a1->real + a2->real;
    result.imag = a1->imag + a2->imag;
    return &result;
}
```
void ComplexAdd(
    struct ComplexNumber *a1,
    struct ComplexNumber *a2,
    struct ComplexNumber *result) {
    result->real = a1->real + a2->real;
    result->imag = a1->imag + a2->imag;
    return;
}
Using It ...

```
struct ComplexNumber j1 = {3.6, 2.125};
struct ComplexNumber j2 = {4.32, 3.1416};
struct ComplexNumber sum;

ComplexAdd(&j1, &j2, &sum);
```
Arrays of structs

```c
struct ComplexNumber j[10];
j[0].real = 8.127649;
j[0].imag = 1.76e18;
```
Arrays, Pointers, and \textit{structs}

/* What's this? */

\begin{verbatim}
struct ComplexNumber *jp[10];
\end{verbatim}

\begin{verbatim}
struct ComplexNumber j0;
jp[0] = &j0;
jp[0]->real = 13.6;
\end{verbatim}
Memory View

jp

j0: 13.6
Quiz 1

```c
struct list_elem {  
    int val;  
    struct list_elem *next;  
} a, b;

int main() {  
    a->val = 1;  
    a->next = &b;  
    b->val = 2;  
    printf("%d\n", a->next->val);  
    return 0;
}
```

- What happens?
  a) syntax error
  b) seg fault
  c) prints something and terminates
struct list_elem {
    int val;
    struct list_elem *next;
} a, b;

int main() {
    a.val = 1;
    a.next = &b;
    b.val = 2;
    printf("%d\n", a.next.val);
    return 0;
}
struct list_elem {
    int val;
    struct list_elem *next;
} a, b;

int main() {
    a.val = 1;
    b.val = 2;
    printf("%d\n", a.next->val);
    return 0;
}
Quiz 4

```c
struct list_elem {
    int val;
    struct list_elem *next;
} a, b;

int main() {
    a.val = 1;
    a.next = &b;
    b.val = 2;
    printf("%d\n", a.next->val);
    return 0;
}
```

- What happens?
  a) syntax error
  b) seg fault
  c) prints something and terminates
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

- **int**
  - Range: 
  - Approximate: 7 decimal digits

- **short**
  - Range: 
  - Approximate: 3 decimal digits

- **char**
  - Range: 
  - Approximate: 0 decimal digits

- **long**
  - Range: 
  - Approximate: 16 decimal digits

- **float**
  - Range: 
  - Approximate: 7 decimal digits

- **double**
  - Range: 
  - Approximate: 16 decimal digits
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

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

def tolower(char c):
    if c >= 'A' and c <= 'Z':
        return c + 'a' - 'A';
    else:
        return c;

Character Strings

```c
char c = 'a';
char *s = "string";
```

![Diagram showing character strings and memory representation]
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";
```

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';
```

```bash
% gcc -o char char.c
% ./char
Segmentation fault
```
Copying Strings (1)

cchar 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));
}
```

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

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

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

```
| a | r | g | 1 | a | r | g | 2 | \0 |
```

```
| a | r | g | 1 | \0 | a | r | g | 2 | \0 |
```

```
| a | r | g | 1 | \0 | a | r | g | 2 | \0 | \0 |
```

```
| a | r | g | 1 | \0 | a | r | g | 2 | \0 | \0 |
```

```
| a | r | g | 1 | \0 | a | r | g | 2 | \0 | \0 |
```

```
| a | r | g | 1 | \0 | a | r | g | 2 | \0 | \0 |
```

```
| a | r | g | 1 | \0 | a | r | g | 2 | \0 | \0 |
```

```
| a | r | g | 1 | \0 | a | r | g | 2 | \0 | \0 |
```

```
| a | r | g | 1 | \0 | a | r | g | 2 | \0 | \0 |
```

```
| a | r | g | 1 | \0 | a | r | g | 2 | \0 | \0 |
```
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 \textit{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;
}
```

\textbf{Output:}

\begin{itemize}
\item arg0
\item arg1
\item arg2
\item arg3
\end{itemize}
char *getfirstword(char **rem_p) {
    char *str = *rem_p;
    if (str == NULL)
        return NULL;
    int len = strlen(str);
    int wlen =
        Strspn(str, " \	\n");
    // initial whitespace
    if (wlen == len) {
        // string is all whitespace
        return NULL;
    }
    str = &str[wlen];
    // skip over whitespace
    len -= wlen;
}

int wlen =
    Strcspn(str, " \	\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;