

CS61B Lecture #29

Administrative: Have a Happy Thanksgiving!

Today:

- Strings
- Generic pointers
- Unions
- What can go wrong.

Strings in C

- Strings are arrays of char in which the last character is ASCII null ('\\000', or just 0).
- Therefore, passed as char* or const char* (not modifiable).
- Somewhat like StringBuffer in Java.
- Library routines for manipulation:

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

```
char A[9];           A: 

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| ? | ? | ? | ? | ? | ? | ? | ? | ? |
|---|---|---|---|---|---|---|---|---|


```

```
char* s = A;
```

```
strcpy(s, "the ");  A: 

|   |   |   |  |     |   |   |   |   |
|---|---|---|--|-----|---|---|---|---|
| t | h | e |  | \\0 | ? | ? | ? | ? |
|---|---|---|--|-----|---|---|---|---|


```

```
/* or */ strcpy(A, "the ");
```

```
strlen(s) == 4
```

```
strcat(s, "text");  A: 

|   |   |   |  |   |   |   |   |     |
|---|---|---|--|---|---|---|---|-----|
| t | h | e |  | t | e | x | t | \\0 |
|---|---|---|--|---|---|---|---|-----|


```

```
strcmp(s, "the end") > 0
```

```
strlen(A) == 8
```

More String Manipulation

A:

t	h	e		t	e	x	t	\0
---	---	---	--	---	---	---	---	----

`strcpy(s, "big");`

A:

b	i	g	\0	t	e	x	t	\0
---	---	---	----	---	---	---	---	----

`strlen(s) == 3`

`s[3] = ' ';`

A:

b	i	g		t	e	x	t	\0
---	---	---	--	---	---	---	---	----

`strlen(s) == 8`

`strcpy(s+4, "hair")`

A:

									?
--	--	--	--	--	--	--	--	--	---

`sprintf(s, "val = %d", 42)`

A:

v	a	l		=		4	2	\0
---	---	---	--	---	--	---	---	----

Generic Pointers

- Java has type `Object`, which can be used as “pointer to anything.”
- C has an idiom with a (somewhat) similar effect:

```
int x;  
void* xp = &x;  
/* *xp is meaningless */  
int y = * (int*) xp; /* Must cast to dereference  
                    * usefully */
```

- Unlike Java, C does not track the type of object `xp` points to.
- No `instanceof`. (Typically) no `ClassCastException` if you get it wrong.

More Generic Values: Unions

- A variable of union type can hold one of a finite set of types of values:

```
typedef union Numeric Numeric;
union Numeric {
    int anInt;
    double aDouble;
    int* anIntPtrtr;
    double* aDoublePntr;
};

Numeric z;
int q;

void f () {
    z.anInt = 5;
    z.aDouble = 3.14; /* Now anInt field undefined */
    z.anIntPtrtr = &q; /* Now aDouble, anInt, undefined */
    ...
}
```

- These unions are *untagged*—no indication of which field is valid.

Many Things Can Go Wrong

- Preceding examples show many opportunities to do something wrong.
- Typically, C implementations don't check for such things, even though the language standard says they're wrong.
- Examples:
 - Indexing off the end of an array.
 - Copying over too large a string with `strcat`
 - Casting `int*` to `void*` and then back to `double*`.
 - Assigning to an `int` field of Numeric type, above, and then fetching from an `IntPtr` field.
 - Using storage after it is freed.
 - Returning pointers to local variables (which vanish during return):

```
char* f () {  
    char answer[128];  
    ... return answer;  
}
```

- Consequences of doing any of these are undefined. C culture is to live with it.