

CS 33

Introduction to C Part 6

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';
```



Works!

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 non-null characters in s1 is now in i
```

Size

```
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  
$
```

Not a Quiz

```
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) {
    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; // s1 > s2
    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 1

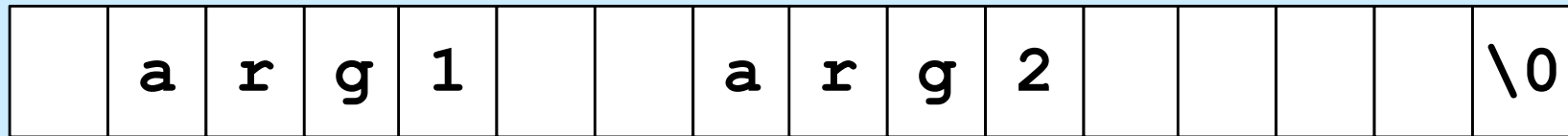
```
#include <stdio.h>
#include <string.h>

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

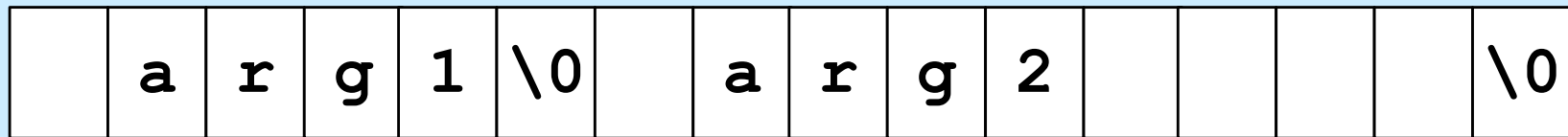
This code:

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

Parsing a String

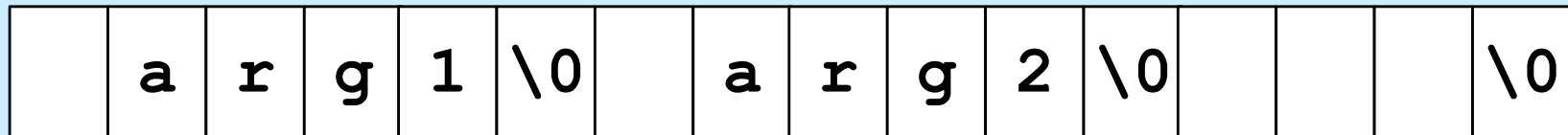


string



token

rem



token

rem

Designing the Parse Function

- **It modifies the string being parsed**
 - puts nulls at the end of each token
- **Each call returns a pointer to the next token**
 - how does it know where it left off the last time?
 - » how is *rem* dealt with?

Design of *strtok*

- **char** **strtok*(**char** **string*,
 const char **sep*)
 - if *string* is non-NULL, *strtok* returns a pointer to the first token in *string*
 - if *string* is NULL, *strtok* returns a pointer to the next token in *string* (the one after that returned in the previous call), or NULL if there are no more tokens
 - tokens are separated by any non-empty combination of characters in *sep*

Using *strtok*

```
int main() {  
    char line[] = "  arg0    arg1 arg2    arg3    ";  
    char *str = line;  
    char *token;  
    while ((token = strtok(str, " \\t\\n")) != NULL) {  
        printf("%s\\n", token);  
        str = NULL;  
    }  
    return 0;  
}
```

Output:

```
arg0  
arg1  
arg2  
arg3
```

strtok Code part 1

```
char *strtok(char *string, const char *sep) {  
    static char *rem = NULL;  
    if (string == NULL) {  
        if (rem == NULL) return NULL;  
        string = rem;  
    }  
    int len = strlen(string);  
    int slen = strspn(str, sep);  
    // initial separators  
    if (slen == len) {  
        // string is all separators  
        rem = NULL;  
        return NULL;  
    }  
}
```


***strtok* Code part 2**

```
string = &string[slen]; // skip over separators
len -= slen;
int tlen = strcspn(string, sep); // length of first token
if (tlen < len) {
    // token ends before end of string: terminate it with 0
    string[tlen] = '\0';
    rem = &string[tlen+1];
} else {
    // there's nothing after this token
    rem = NULL;
}
return string;
}
```

Numeric Conversions

```
short a;
```

```
int b;
```

```
float c;
```

```
b = a;    /* always works */
```

```
a = b;    /* sometimes works */
```

```
c = b;    /* sort of works */
```

```
b = c;    /* sometimes works */
```

Implicit Conversions (1)

```
float x, y=2.0;
```

```
int i=1, j=2;
```

```
x = i/j + y;
```

```
/* what's the value of x? */
```

Implicit Conversions (2)

```
float x, y=2.0;
```

```
int i=1, j=2;
```

```
float a, b;
```

```
a = i;
```

```
b = j;
```

```
x = a/b + y;
```

```
/* now what's the value of x? */
```

Explicit Conversions: Casts

```
float x, y=2.0;
```

```
int i=1, j=2;
```

```
x = (float)i / (float)j + y;
```

```
/* and now what's the value of x? */
```

Purposes of Casts

- **Coercion**

```
int i, j;  
float a;  
a = (float) i / (float) j;
```

modify the
value
appropriately

- **Intimidation**

```
float x, y;  
// sizeof(float) == 4  
swap((int *) &x, (int *) &y);
```

it's ok as is
(trust me!)

Quiz 2

- Will this work?

```
double x, y; //sizeof(double) == 8
```

```
...
```

```
swap ( (int *) &x, (int *) &y );
```

- a) yes
- b) no

Caveat Emptor

- **Casts tell the C compiler:**
“Shut up, I know what I’m doing!”

- **Sometimes true**

```
float x, y;  
swap( (int *) &x, (int *) &y );
```

- **Sometimes false**

```
double x, y;  
swap( (int *) &x, (int *) &y );
```


Nothing, and More ...

- ***void* means, literally, nothing:**

```
void NotMuch(void) {  
    printf("I return nothing\n");  
}
```

- **What does *void ** mean?**
 - it's a pointer to anything you feel like
 - » a generic pointer

Rules

- **Use with other pointers**

```
int *x;  
void *y;  
x = y; /* legal */  
y = x; /* legal */
```

- **Dereferencing**

```
void *z;  
func(*z); /* illegal! */  
func(*(int *)z); /* legal */
```

Swap, Revisited

```
void swap(int *i, int *j) {  
    int tmp;  
    tmp = *j; *j = *i; *i = tmp;  
}  
/* can we make this generic? */
```

An Application: Generic Swap

```
void gswap (void *p1, void *p2,  
            int size) {  
    int i;  
    for (i=0; i < size; i++) {  
        char tmp;  
        tmp = ((char *)p1)[i];  
        ((char *)p1)[i] = ((char *)p2)[i];  
        ((char *)p2)[i] = tmp;  
    }  
}
```

Using Generic Swap

```
short a=1, b=2;  
gswap(&a, &b, sizeof(short));
```

```
int x=6, y=7;  
gswap(&x, &y, sizeof(int));
```

```
int A[] = {1, 2, 3}, B[] = {7, 8, 9};  
gswap(A, B, sizeof(A));
```

Fun with Functions (1)

```
void ArrayDouble(int A[], int len) {  
    int i;  
    for (i=0; i<len; i++)  
        A[i] = 2*A[i];  
}
```

Fun with Functions (2)

```
void ArrayBop(int A[],  
             int len,  
             int (*func) (int)) {  
    int i;  
    for (i=0; i<len; i++)  
        A[i] = (*func) (A[i]);  
}
```

Fun with Functions (3)

```
int triple(int arg) {  
    return 3*arg;  
}
```

```
int main() {  
    int A[20];  
    ... /* initialize A */  
    ArrayBop(A, 20, triple);  
    return 0;  
}
```


Laziness ...

- Why type the declaration

```
void * (*f) (void *, void *);
```

- You could, instead, type

```
MyType f;
```

- (If, of course, you can somehow define *MyType* to mean the right thing)

typedef

- **Allows one to create new names for existing types**

```
typedef int *IntP_t;
```

```
IntP_t x;
```

– means the same as

```
int *x;
```

More typedefs

```
typedef struct complex {  
    float real;  
    float imag;  
} complex_t;
```

```
complex_t i, *ip;
```

And ...

```
typedef void * (*MyFunc_t) (void *, void *);
```

```
MyFunc_t f;
```

```
// you must do its definition the long way
```

```
void *f(void *a1, void *a2) {  
    ...  
}
```

Quiz 3

- What's A?

```
typedef double X_t[M];  
X_t A[N];
```

- a) an array of N doubles
- b) an MxN array of doubles
- c) an NxM array of doubles
- d) a syntax error