Personal SE

Functions, Arrays, Strings & Command Line Arguments
Functions in C

• Syntax like Java methods but w/o public, abstract, etc.
• As in Java, all arguments (well, most arguments) are passed by value.
• Example:

```c
void try_swap( int x, int y ) {
    int t = x ;
    x = y ;
    y = t ;
}
```
• Doesn't work:
  – x and y are copies of the arguments in the caller.
  – Changing the copy has no effect in the caller.
Functions in C

• Functions must be declared before use:
  – *Declare* means specify name, return value, and argument types.
• Indeed, in C *everything* must be declared before use!
Functions in C

• Functions must be declared before use:
  – *Declare* means specify name, return value, and argument types.
• Indeed, in C *everything* must be declared before use!

extern int min(int x, int y);  // Declaration of min
static int max(int x, int y);  // Declaration of max

int max_div_min(int x, int y) {
    return max(x, y) / min(x, y);
}

int min(int x, int y) {     // Definition of min
    return (x <= y) ? x : y;
}

static int max(int x, int y) {      // Definition of max
    return (x >= y) ? x : y;
}
Functions in C

• Functions must be declared before use:
  – *Declare* means specify name, return value, and argument types.
• Indeed, in C *everything* must be declared before use!

```c
extern int min(int x, int y);  // Declaration of min
static int max(int x, int y);  // Declaration of max
int max_div_min(int x, int y) {
    return max(x, y) / min(x, y) ;
}
int min(int x, int y) {              // Definition of min
    return (x <= y) ? x : y ;
}
static int max(int x, int y) {      // Definition of max
    return (x >= y) ? x : y ;
}
```
Functions in C

• Functions must be declared before use:
  – *Declare* means specify name, return value, and argument types.
• Indeed, in C *everything* must be declared before use!

```c
extern int min(int x, int y);  // Declaration of min
static int max(int x, int y);  // Declaration of max
int max_div_min(int x, int y) {
    return max(x, y) / min(x, y);
}
int min(int x, int y) {       // Definition of min
    return (x <= y) ? x : y;
}
static int max(int x, int y) {  // Definition of max
    return (x >= y) ? x : y;
}
```
Arrays in C

- Generic form: `type name[size]` ;
- Examples:

```c
#define MAX_SAMPLES (100)
int samples[MAX_SAMPLES] ;
```
Arrays in C

• Generic form: `type name[size] ;`

• Examples:

```c
#define MAX_SAMPLES (100)
int samples[MAX_SAMPLES] ;
```

Array of 100 integers. Indices run 0 .. 99
NO SUBSCRIPT CHECKS!
NOTE THE USE OF SYMBOLIC CONSTANT!
Arrays in C

- Generic form: `type name[size]`;
- Examples:

```c
#define MAX_SAMPLES (100)  
int samples[MAX_SAMPLES] ;  

int sum = 0 ;  
int i ;  

for ( i = 0 ; i < MAX_SAMPLES ; ++i ) {  
    sum += samples[i] ;  
}
```

Simple summation of array values.
#define DIMENSION (50) ;
double m1[DIMENSION][DIMENSION] ;
Arrays in C

#define DIMENSION (50);
double m1[DIMENSION][DIMENSION];

A matrix or a 2 dimensional array
Access by m1[i][j]
#define DIMENSION (50) 
double m1[DIMENSION][DIMENSION] ;
double m2[DIMENSION][DIMENSION] ;
double product[DIMENSION][DIMENSION] ;

int i, j, k ;

for ( i = 0 ; i < DIMENSION ; ++i ) {
    for ( j = 0 ; j < DIMENSION ; ++j ) {
        product[ i ][ j ] = 0.0 ;
        for ( k = 0 ; k < DIMENSION ; ++k ) {
            product[i][j] += m1[i][k] * m2[k][j] ;
        }
    }
}
Arrays in C

- Arrays are passed to functions by reference.
Arrays in C

• Arrays are passed to functions by *reference*.

• Changes to the array contents in the function will be visible to the caller, e.g., array copy.
Arrays in C

- Arrays are passed to functions by reference.
- Changes to the array contents in the function will be visible to the caller, e.g., array copy.

```c
void acopy( int to[], int from[], size ) {
    int i ;

    for( i = 0 ; i < size ; i++ ) {
        to[ i ] = from[ i ] ;
    }
}
```
Arrays in C

- Arrays are passed to functions by reference.
- Changes to the array contents in the function will be visible to the caller, e.g., array copy.

```c
void acopy( int to[], int from[], size ) {
    int i ;
    for( i = 0 ; i < size ; i++ ) {
        to[ i ] = from[ i ];
    }
}
```

Need not, but may, give the array size.
Arrays in C - Review

• Array size fixed at definition time.

• Good practice (that is, **OUR** practice) is to use symbolic constants to define array sizes.

• Array indices are integers.

• Legal indices run from 0 to $arraysize - 1$

• C will **not** prevent you from indexing outside the bounds of the array (no subscript checks).

• Arrays are passed by reference.
Strings in C

• A string is just an array of chars:

```c
char welcome[] = "Hello" ;  // C permits this
```

• This is an array of 8-bit bytes holding ASCII characters.
Strings in C

• A string is just an array of chars:
  char welcome[] = "Hello" ;  // C permits this

• This is an array of 8-bit bytes holding ASCII characters

• The array in memory looks like this:
  ['H', 'e', 'l', 'l', 'o', '\0']

• Whoa! What’s that last character?????
Strings in C

• A string is just an array of chars:
  ```c
  char welcome[] = "Hello" ;  // C permits this
  ```
• This is an array of 8-bit bytes holding ASCII characters
• The array in memory looks like this:
  ```
  'H' 'e' 'l' 'l' 'o' NUL
  ```
• Whoa! What’s that last character????
• In C, proper strings must be terminated with a NUL (0) character.
• We always need an extra byte to hold the terminator!
Strings in C

• Assume we are reading and processing lines of text, where at most the first 80 characters of a line are useful
• How would we declare an array to hold the line as a string?
Strings in C

• Assume we are reading and processing lines of text, where at most the first 80 characters of a line are useful

• How would we declare an array to hold the line as a string?

```c
#define MAXLINE (80)
char line[ MAXLINE + 1 ] ;    // 1 extra character for the NUL
```
Strings in C

- Assume we are reading and processing lines of text, where at most the first 80 characters of a line are useful.
- How would we declare an array to hold the line as a string?
  
  ```c
  #define MAXLINE (80)
  char line[ MAXLINE + 1 ] ;    // 1 extra character for the NUL
  ```
- How would we read in such a line?
Strings in C

• Assume we are reading and processing lines of text, where at most the first 80 characters of a line are useful.

• How would we declare an array to hold the line as a string?

```c
#define MAXLINE (80)
char line[ MAXLINE + 1 ] ;    // 1 extra character for the NUL
```

• How would we read in such a line?

```c
void readline( char line[], int maxsize ) {
    int i = 0 ;
    int ch ;
    for ( ch = getchar() ; ch != '\n' && ch != EOF ; ch = getchar() ) {
        if ( i < maxsize ) {
            line[ i++ ] = ch ;
        }
    }
    line[ i ] = '\0' ;
    return ;
}
```
Strings in C

- How can we copy one string to another?
- Modify acopy to strcpy:

```c
void strcpy( char sto[], char sfrom[] ) {
    int i;

    for ( i = 0 ; sto[ i ] = sfrom[ i ] ; ++i )
        ;
}
```
Strings in C

- How can we copy one string to another?
- Modify acopy to strcpy:

```c
void strcpy( char sto[], char sfrom[] ) {
    int i ;
    for ( i = 0 ; sto[ i ] = sfrom[ i ] ; ++i ) ;
}
```

Copy the \( i \)th character. If this was a NUL, exit the loop.
#include <string.h>

int strlen( char str[] ) ;

Note: strlen("Hello") == 5

void strcpy( char sto[], char sfrom[] ) ;

void strncpy( char sto[], char sfrom[], unsigned n ) ;

Note: Copies 'n' characters to 'sto' from 'sfrom', padding with '\0' as necessary.

Note: If 'sfrom' is too long to fit in 'sto', then 'sto' will NOT be NUL terminated.

int strcmp( char str1[], char str2[] ) ;

Note: comparison is in dictionary order.

Note: returns -1, 0, 1 if 'str1' is less than, equal to, or greater than 'str2', respectively.
The full declaration of main is:

```c
int main( int ac, char **argv ) ;
```
The full declaration of main is:

```c
int main(int ac, char **argv);
```

ac = argument count (the number of command line arguments).

ac >= 1, as the program name is the 0th argument.
Command Line Arguments

The full declaration of main is:

```c
int main(int ac, char **argv);
```

ac = argument count (the number of command line arguments).
Includes the program name as the 0th argument.

Example: ac == 5
```
gcc  -o  myprog  main.c  util.c
```
0     1     2     3     4
The full declaration of main is:

```c
int main( int ac, char **argv ) ;
```

`argv` = the argument vector - allows access to the arguments

  it's a pointer, but don't worry - treat it like a 2D array.

  `argv[ i ]` is the `i`th argument as a string (array).

  `argv[ i ][ j ]` is the `j`th character of the `i`th argument.
```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main( int ac, char **argv ) {
    int i ;

    printf( "Program name = %s\n", argv[0] ) ;

    for( i = 1 ; i < ac ; ++i ) {
        printf( "argv[%d] = %s ", i, argv[i] ) ;
        printf( "and its length is %d\n", strlen( argv[i] ) ) ;
    }

    return 0 ;
}
```