Personal SE

Functions & Arrays
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:
  - \textit{x} and \textit{y} are copies of the arguments in the caller.
  - Changing the copy has \textit{no effect} in the caller.
Functions in C

• Functions must be declared before use:
  – *Declare* means specify name, return value, and argument types.
  – Technically functions can default to an implicit declaration. Never rely on implicit declaration!

• 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!

```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;
}
```
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;
}
```

*static*: defined and known only in this C source file.
Arrays in C

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

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

• Generic form: \( \text{type name}[\text{size}] \);

• 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: \textit{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.
Arrays in C

#define DIMENSION (50) ;
double m1[DIMENSION][DIMENSION] ;
Arrays in C

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

A matrix or a 2 dimensional array
Access by `m1[i][j]`
Arrays in C

```c
#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] ;
        }
    }
}
```

Matrix multiplication to show use of double indices.
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 \textit{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[], int 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[], int 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.