CSE 230
Intermediate Programming in C and C++

The Preprocessor

Fall 2017

Stony Brook University

Instructor: **Shebuti Rayana**

http://www3.cs.stonybrook.edu/~cse230/

Introduction

- C uses preprocessor to extend its power and notations, e.g., `#include`, `#define`
- `#define` macro: can be used to generate inline code that takes the place of a function call.
  - can reduce program execution time.
- Lines that begin with a # are called preprocessing directives.
  - These lines communicate with the preprocessor.
- The effect of a preprocessing directive starts at its place in a file and continues until the end of that file, or until its effect is negated by another directive.

The use of **`#include`**

- Preprocessing directive:
  ```
  #include <stdio.h>
  #include <stdlib.h>
  ```
  – This causes the preprocessor to replace the line with a copy of the contents of the named file.

  – The preprocessor looks for the file only in the other places and not in the current directory.

  – In UNIX systems, the standard header files such as stdio.h and stdlib.h are typically found in /usr/include
- Another form: `#include “filename”`
  – A search for the file is made first in the current directory and then in other system-dependent places.

The use of **`#include`** (cont.)

- There is no restriction on what a `#include` file can contain.
- It can contain other preprocessing directives that will be expanded by the preprocessor in turn.

The use of **`#define`**

- Occur in two forms:

  `#define identifier token_string`$_{opt}$
  `#define identifier(identifier`$_{1}$`, …, \`
  `identifier`$_{n}$`) token_string`$_{opt}$

  - The `token_string` is optional.
  - A long definition can be continued to the next line by placing a backslash \ at the end of the current line.
  - If a simple `#define` of the first form occurs in a file, the preprocessor replaces every occurrence of `identifier` by `token_string` in the remainder of the file, except in quoted strings.
  - The use of simple `#define` can improve program clarity and portability.
  -

Example: #define

- `#define SECONDS_PER_DAY  (60*60*24)`
- `#define PI 3.14159`
- `#define c 299792.458 //speed of light km/s`
- `#define EOF (-1) //end-of-file marker`
- `#define MAXINT 2147483647`
- `#define ITERS 50 //number of iterations`
- `#define SIZE 250 //array size`
- `#define EPS 1.0e-9 //numerical constant`
- `#define EQ ==`

Macros with Arguments

No space

- General form:
  #define identifier(identifier$_1$, …,\
  identifier$_n$) token_string$_{opt}$
  - Can have zero or more parameters
- Example: `#define SQ(x) ((x)*(x))`
  - With argument 7 + w:
    SQ(7 + w) expands to ((7 + w)*(7 + w))
  - Similarly,
    SQ(SQ(*p)) expands to ((((*p)*(*p))) * (((*p)*(*p))))
- This seemingly extravagant use of parentheses is to protect against the macro expanding an expression so that it led to an unanticipated order of evaluation.

Macros with Arguments (cont.)

- Why all the parenthesis are important?

1. Suppose we have: `#define SQ(x)   x*x`

– Then for a + b:
SQ(a + b) expands to a + b * a + b which is not same as
((a + b) * (a + b))

2. `#define SQ(x)  (x)  *  (x)`

– 4 / SQ(2) expands to 4 / (2) * (2) which is not same as
4 / ((2) * (2))

3. `#define SQ   (x)    ((x)*(x))`

– SQ(7) expands to (x) ((x)*(x))(7) /* wrong */

A common mistake with `#define`

- Putting semicolon at the end of #define
- `#define  SQ(x) ((x) * ((x)); /* error */`
- `x = SQ(y);` gets expanded to `x = ((y) * (y));;`
  - Creates an unwanted null statement

-
  ```
  if (x == 2)
    x = SQ(y);
  else
    ++x;
  ```
- The extra semicolon does not allow the else to be attached to the if statement.

Macros as Function call

- Instead of writing a function to find the minimum of two values, a programmer could write

```
#define min(x,y) (((x) < (y)) ? (x) : (y))
```

- `m = min(u,v)` expand to

```
m = (((u) < (v)) ? (u) : (v))
```

  – The arguments of min() can be arbitrary expressions of compatible type.

- We can use `min()` to define another macro,

```
#define min4(a,b,c,d) \
min(min(a,b),min(c,d))
```

Macros as Function call (cont.)

- A macro definition can use both functions and macros in its body.

```
#define SQ(x) ((x) * (x))

#define CUBE(x) (SQ(x) * (x))

#define F_POW(x) sqrt(sqrt(CUBE(x)))
/* fractional power:3/4 */
```

Use of `#undef`

- A preprocessing directive of the form
  `#undef identifier`
  - will undefine a macro.
  - It causes the previous definition of a macro to be forgotten.

Type Definition

- C provides the `typedef` facility so that an identifier can be associated with a specific type.
- Example: `typedef char uppercase;`
  - This makes uppercase a type that is synonymous with char, and it can be used in declarations

```
uppercase c, u[1100];
```

Type Definition and Macros in `stddef.h`

- typedef int ptrdif_t; /* pointer difference type */
  - The type ptrdif_t tells what type is obtained with an expression involving the difference of two pointers.
- typedef short wchar_t; /* wide character type */
  - The type wchar_t is provided to support languages with character sets that will not fit into a char.
- typedef unsigned size_t; /* the sizeof type */
- The macro NULL is also given in stddef.h. It is an implementation-defined null pointer constant.
  - NULL is defined to be 0, but on some systems it is given by #define NULL ((void *) 0)

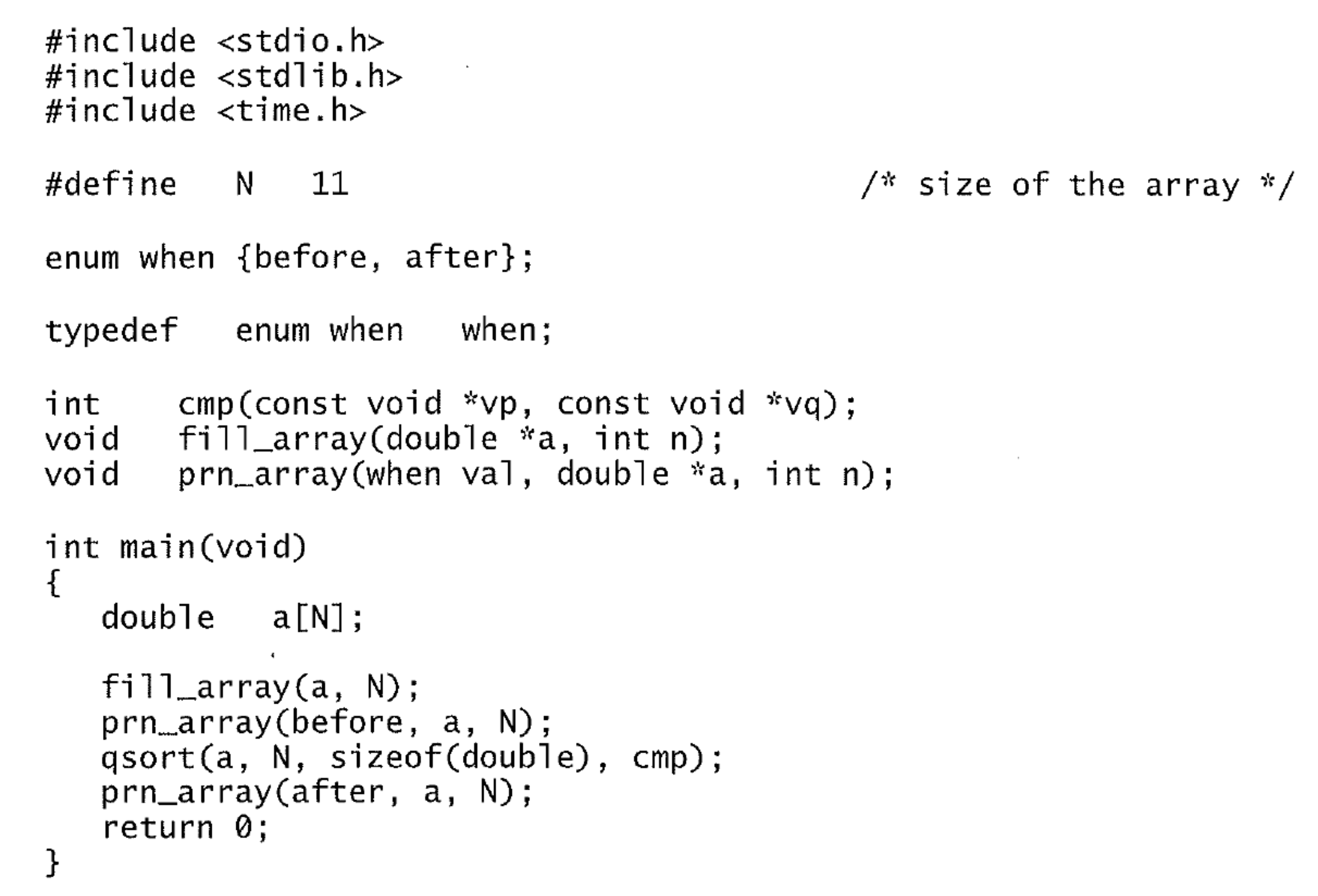
Example: qsort()

- Function prototype of quicksort in stdlib.h

  ```
  void qsort(void * array, size_t
  n_els, size_t el_size, int
  compare(const void*, const void*))
  ```

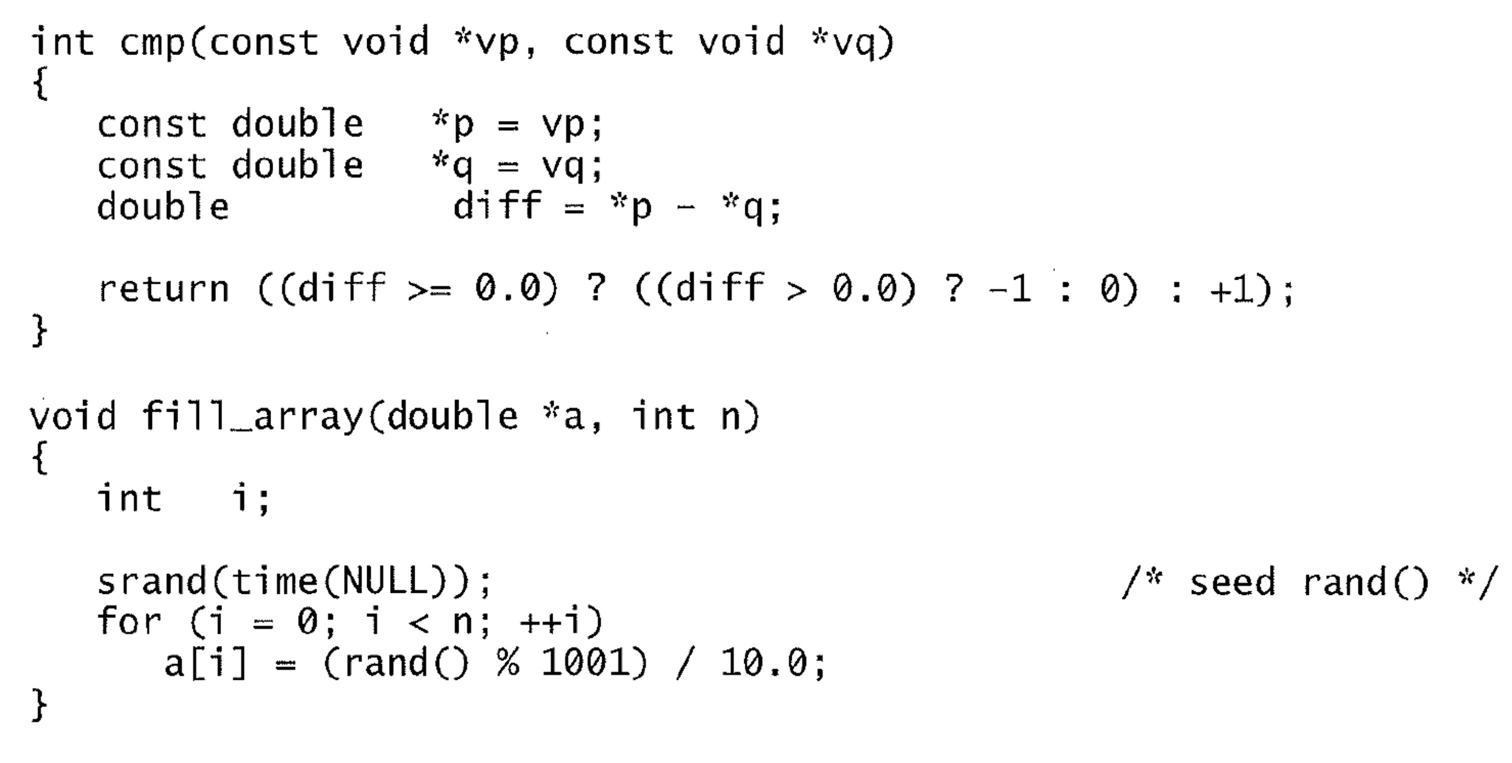
- The comparison function returns an int that is less than, equal to, or greater than zero, depending on whether its first argument is considered to be less than, equal to, or greater than its second argument.

Test program for qsort()



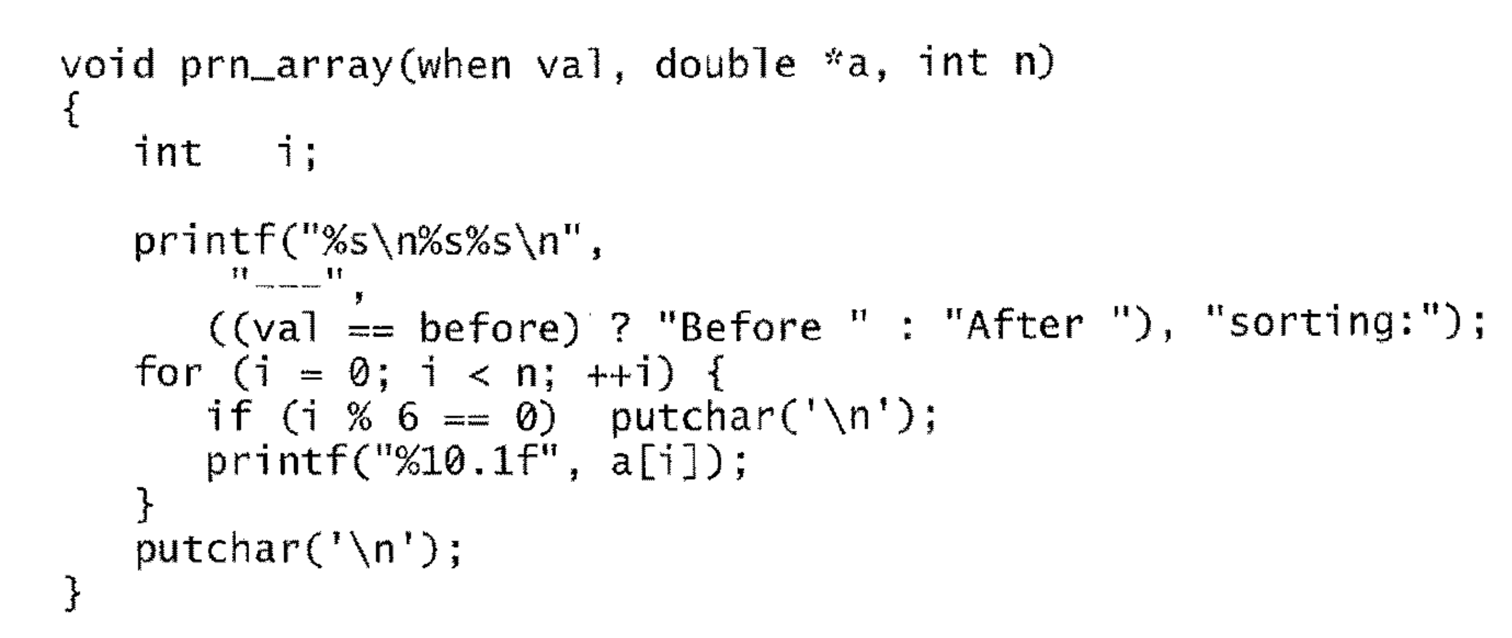
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define   N   11                              /* size of the array */

enum when {before, after};

typedef   enum when   when;

int    cmp(const void *vp, const void *vq);
void   fill_array(double *a, int n);
void   prn_array(when val, double *a, int n);

int main(void)
{
   double   a[N];

   fill_array(a, N);
   prn_array(before, a, N);
   qsort(a, N, sizeof(double), cmp);
   prn_array(after, a, N);
   return 0;
}
```

Test program for qsort() (cont.)

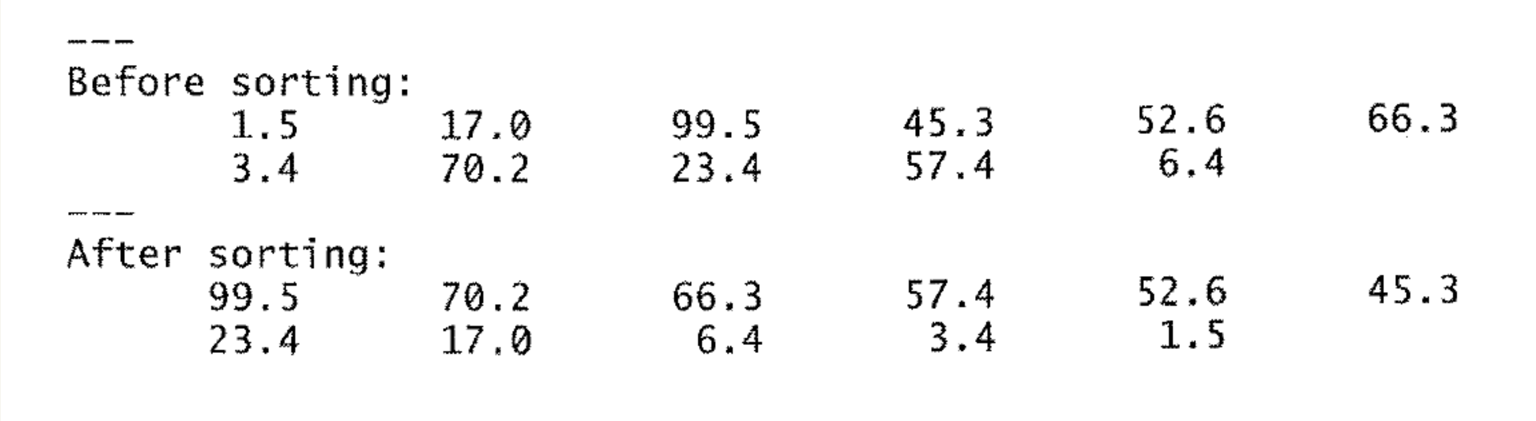


```
int cmp(const void *vp, const void *vq)
{
   const double   *p = vp;
   const double   *q = vq;
   double          diff = *p - *q;

   return ((diff >= 0.0) ? ((diff > 0.0) ? -1 : 0) : +1);
}

void fill_array(double *a, int n)
{
   int   i;

   srand(time(NULL));                              /* seed rand() */
   for (i = 0; i < n; ++i)
      a[i] = (rand() % 1001) / 10.0;
}
```

Test program for qsort() (cont.)

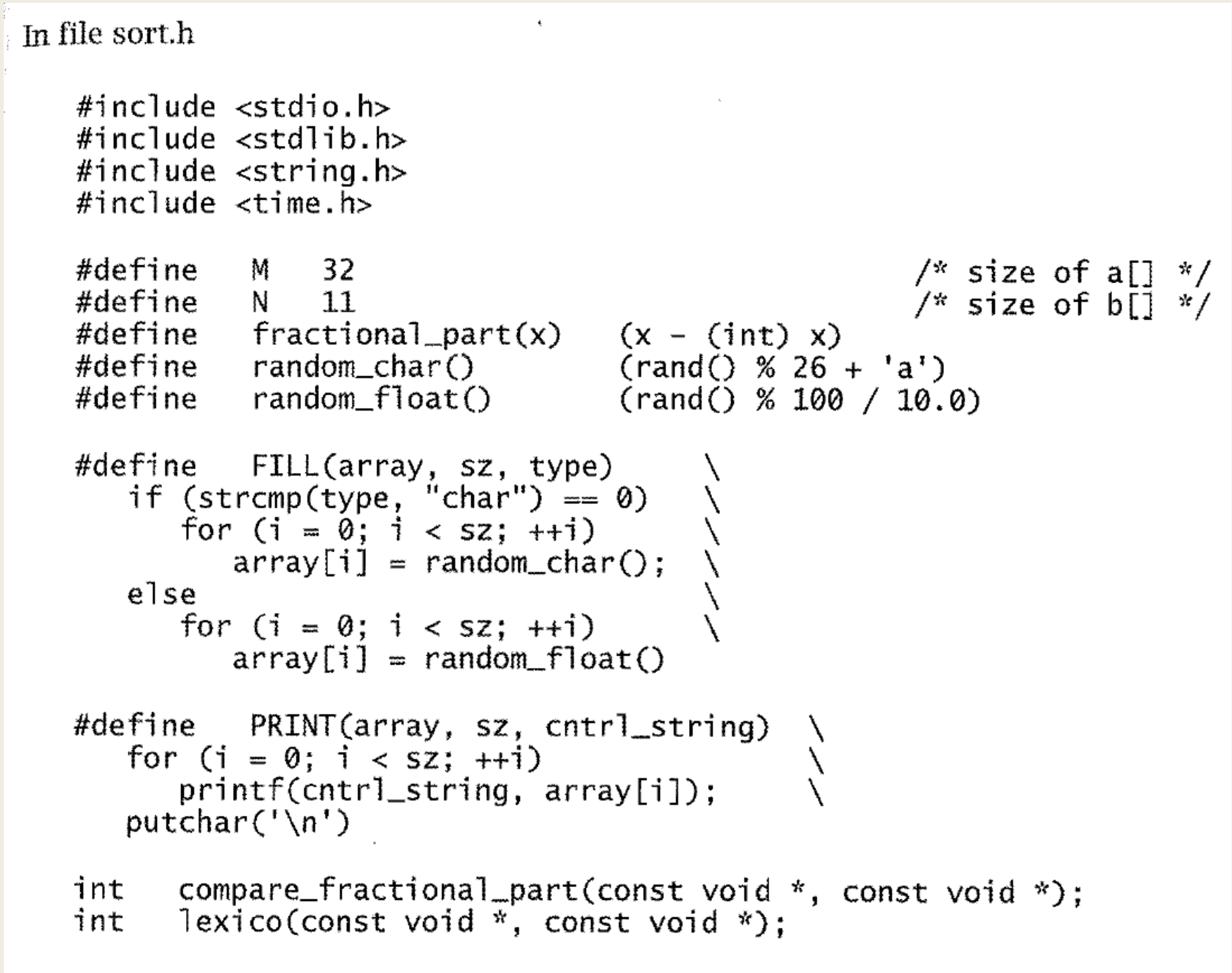


```
void prn_array(when val, double *a, int n)
{
   int   i;

   printf("%s\n%s%s\n",
       "---",
      ((val == before) ? "Before " : "After "), "sorting:");
   for (i = 0; i < n; ++i) {
      if (i % 6 == 0)  putchar('\n');
      printf("%10.1f", a[i]);
   }
   putchar('\n');
}
```



```
---
Before sorting:
       1.5      17.0      99.5      45.3      52.6      66.3
       3.4      70.2      23.4      57.4       6.4
---
After sorting:
      99.5      70.2      66.3      57.4      52.6      45.3
      23.4      17.0       6.4       3.4       1.5
```

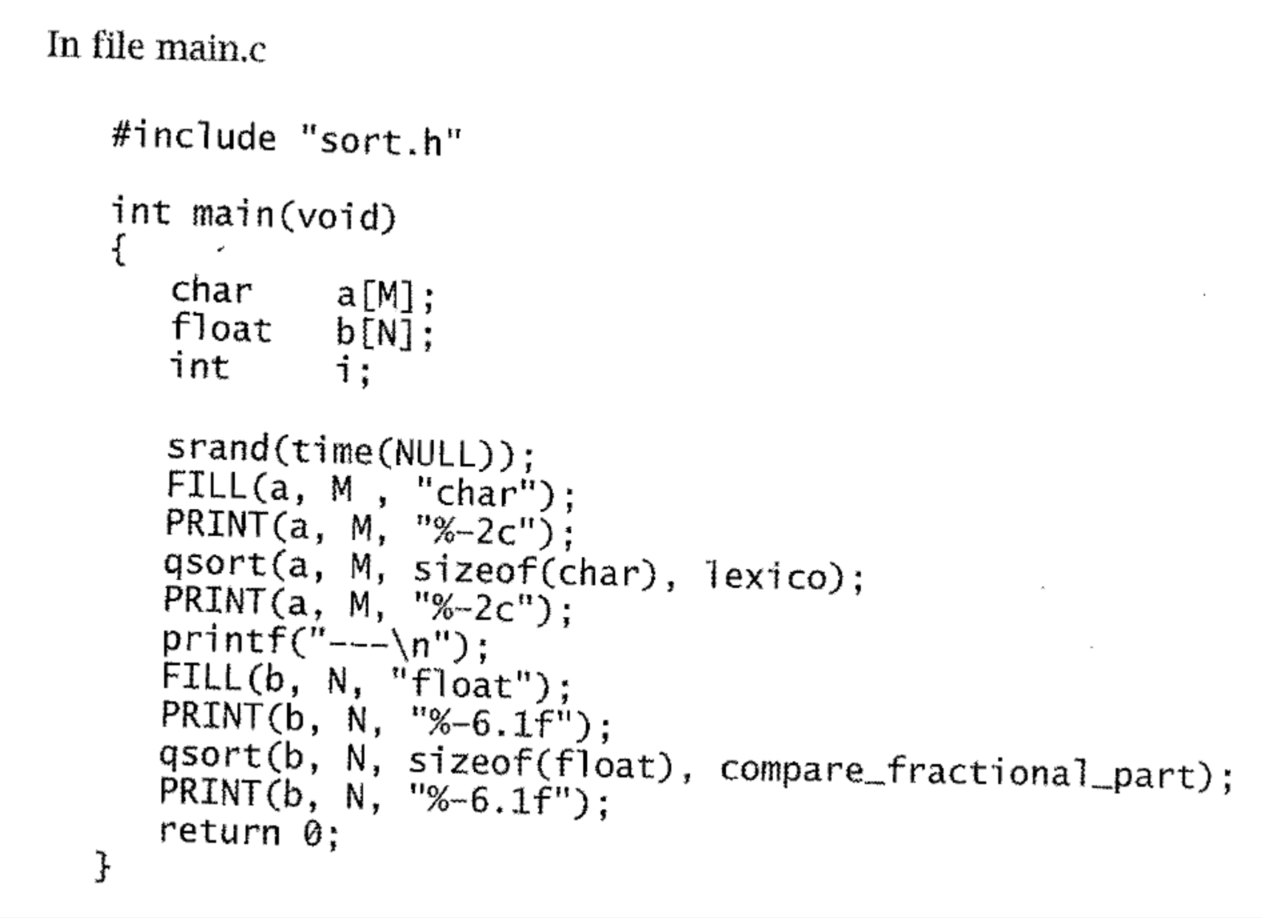
Test code qsort() with macros



In file sort.h

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

#define   M   32                                        /* size of a[] */
#define   N   11                                        /* size of b[] */
#define   fractional_part(x)    (x - (int) x)
#define   random_char()         (rand() % 26 + 'a')
#define   random_float()        (rand() % 100 / 10.0)

#define   FILL(array, sz, type)       \
   if (strcmp(type, "char") == 0)     \
      for (i = 0; i < sz; ++i)        \
         array[i] = random_char();    \
   else                               \
      for (i = 0; i < sz; ++i)        \
         array[i] = random_float()

#define   PRINT(array, sz, cntrl_string)  \
   for (i = 0; i < sz; ++i)               \
      printf(cntrl_string, array[i]);     \
   putchar('\n')

int   compare_fractional_part(const void *, const void *);
int   lexico(const void *, const void *);
```

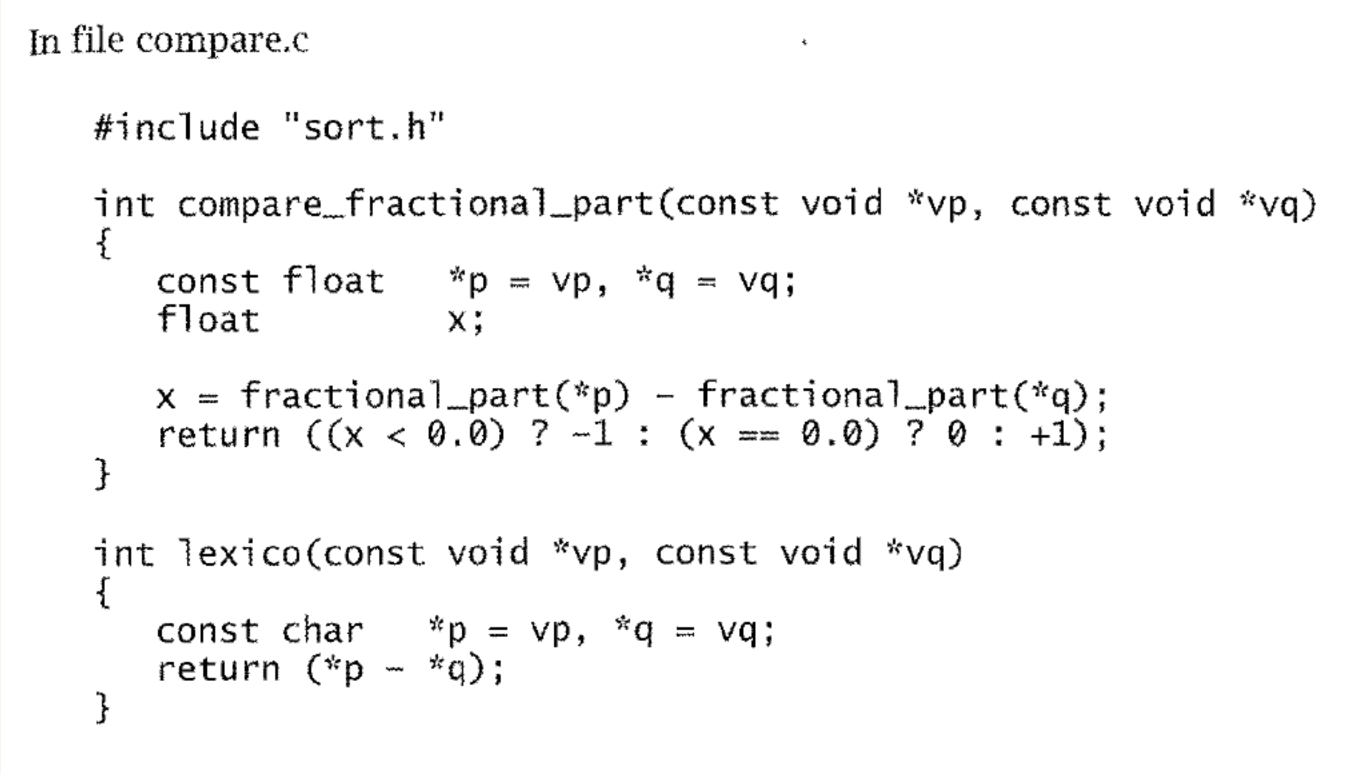
Test code qsort() with macros (cont.)



In file main.c

```
#include "sort.h"

int main(void)
{
    char     a[M];
    float    b[N];
    int      i;

    srand(time(NULL));
    FILL(a, M , "char");
    PRINT(a, M, "%-2c");
    qsort(a, M, sizeof(char), lexico);
    PRINT(a, M, "%-2c");
    printf("---\n");
    FILL(b, N, "float");
    PRINT(b, N, "%-6.1f");
    qsort(b, N, sizeof(float), compare_fractional_part);
    PRINT(b, N, "%-6.1f");
    return 0;
}
```

Test code qsort() with macros (cont.)



In file compare.c

```
#include "sort.h"

int compare_fractional_part(const void *vp, const void *vq)
{
   const float   *p = vp, *q = vq;
   float         x;

   x = fractional_part(*p) - fractional_part(*q);
   return ((x < 0.0) ? -1 : (x == 0.0) ? 0 : +1);
}

int lexico(const void *vp, const void *vq)
{
   const char   *p = vp, *q = vq;
   return (*p - *q);
}
```

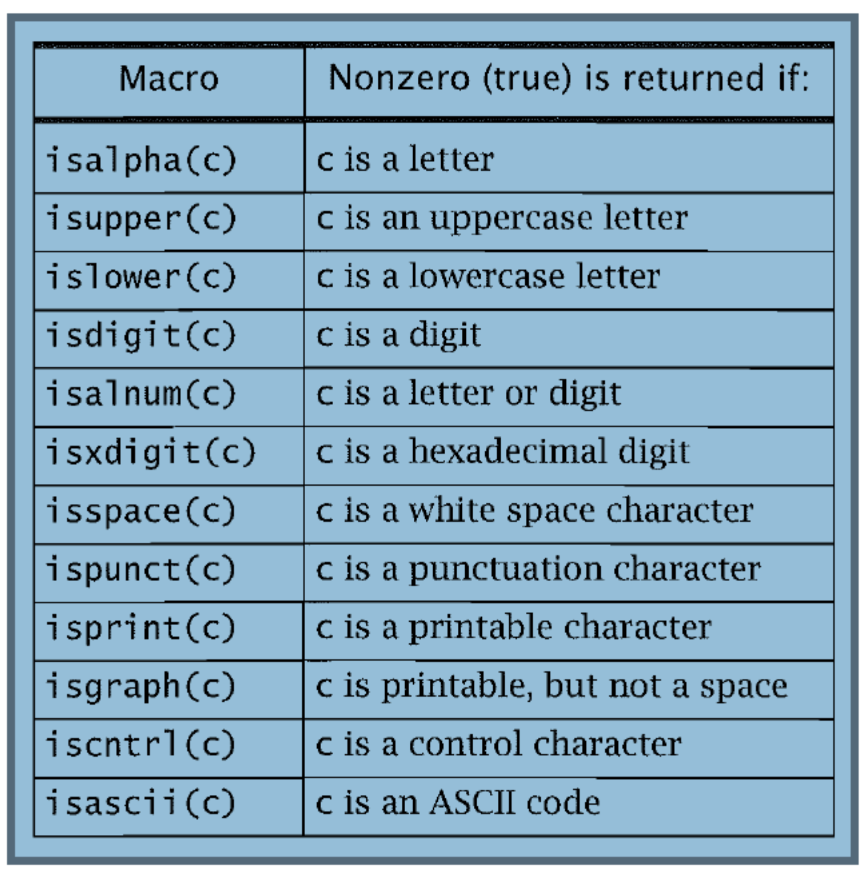
Macros in stdio.h and ctype.h

- Macros getc() and putc() are in stdio.h.
  - (i) read a character from a file, (ii) write a character to a file.

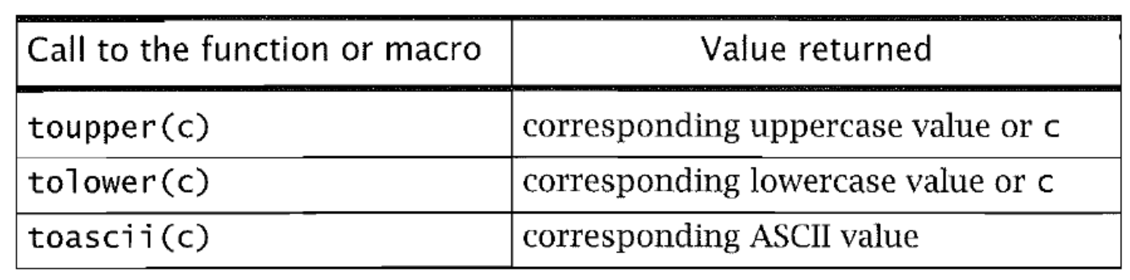
```
#define getchar()\
getc(stdin)
```

```
#define putchar(c)\
putc((c), stdout)
```

  - (i) read characters from the keyboard (ii) write characters to the screen



Macro	Nonzero (true) is returned if:
`isalpha(c)`	c is a letter
`isupper(c)`	c is an uppercase letter
`islower(c)`	c is a lowercase letter
`isdigit(c)`	c is a digit
`isalnum(c)`	c is a letter or digit
`isxdigit(c)`	c is a hexadecimal digit
`isspace(c)`	c is a white space character
`ispunct(c)`	c is a punctuation character
`isprint(c)`	c is a printable character
`isgraph(c)`	c is printable, but not a space
`iscntrl(c)`	c is a control character
`isascii(c)`	c is an ASCII code



Call to the function or macro	Value returned
`toupper(c)`	corresponding uppercase value or `c`
`tolower(c)`	corresponding lowercase value or `c`
`toascii(c)`	corresponding ASCII value

Conditional Compilation

- The preprocessor has directives for conditional compilation.
  - They can be used for program development and for writing code that is more easily portable from one machine to another.

```
#if constanl_integral_expression

#ifdef identifier

#ifndef identifier
```

provides for conditional compilation of the code that follows until the preprocessing directive

- `#endif` is reached. For the intervening code to be compiled, after #if the constant expression must be nonzero (true), and after #ifdef or after #ifndef, the named identifier must have been defined previously in a #define line, without an intervening #undef identifier having been used to undefine the macro.

Example: Conditional Compilation

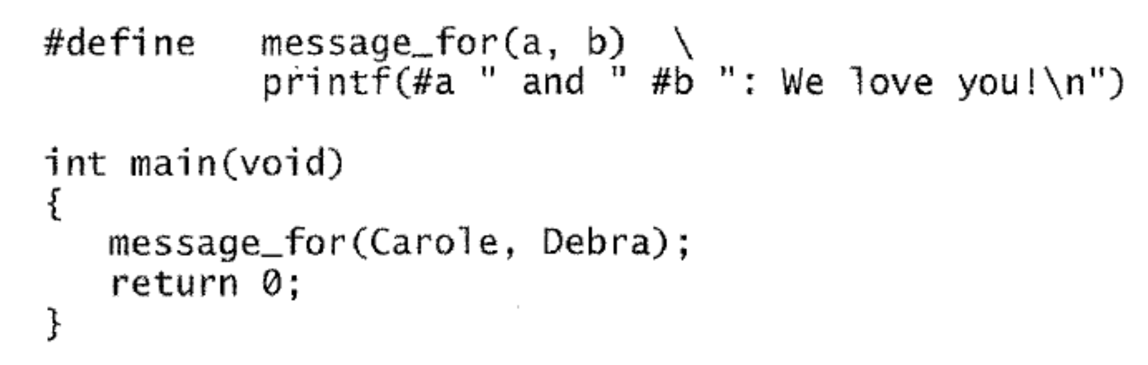
- Sometimes printf() statements are useful for debugging purposes. Suppose that at the top of a file we write `#define DEBUG 1`
- and then throughout the rest of the file we write lines such as

```
#if DEBUG
printf("debug: a = %d\n”, a);
#endif
```

The Predefined Macros

Predefined macro	Value
__DATE__	A string containing the current date
__FILE__	A string containing the file name
__LINE__	An integer representing the current line number
__STDC__	If the implementation follows ANSI Standard C, then the value is a nonzero integer,
__TIME__	A string containing the current time

Operators # and



```
#define   message_for(a, b)  \
          printf(#a " and " #b ": We love you!\n")

int main(void)
{
   message_for(Carole, Debra);
   return 0;
}
```

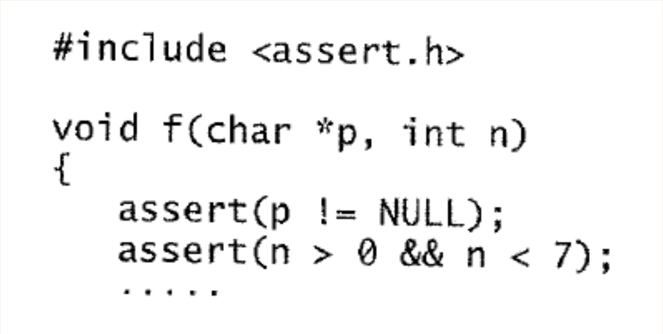
- Unary # causes arguments to be surrounded by double quote
- Binary ## used to merge tokens

```
#define X(i) x ## i
```

`X(1) = X(2) = X(3);` expand to
`x1 = x2 = x3;`

The assert() Macro

- This macro in the standard header file assert.h
- This macro can be used to ensure that the value of an expression is what you expect it to be.
- Suppose that you are writing a critical function and that you want to be sure the arguments satisfy certain conditions.
- If an assertion fails, then the system will print out a message and abort the program.



```
#include <assert.h>

void f(char *p, int n)
{
    assert(p != NULL);
    assert(n > 0 && n < 7);
    .....
```

Use of #error

```
#if A_SIZE < B_SIZE
   #error "Incompatible sizes"
#endif
```

- If during compilation the preprocessor reaches the #error directive, then a compile time error will occur, and the string following the directive will be printed on the screen.