Preprocessors

Preprocessor

- The C language uses the preprocessor to expand its power and notation
- *preprocessor directives*
  - lines beginning with a #
  - communicates with the preprocessor

#include

#define

Use of **#include**

#include <stdio.h>

#include <stdlib.h>

- The preprocessor looks for the file only in the places where standard header files are stored, not in the current directory.
- Where the standard header files are stored is system-dependent.

#include “filename”

- Search is made first in the current directory,
- and then in other system-dependent places

Use of **#define** (1/2)

#define *identifier token_string*

- The preprocessor replaces every occurrence of ***identifier*** by ***token_string*** in the remainder of the file, except in quoted string.
- ***token_string*** is optional

```
#define  SECONDS_PER_DAY  (60*60*24)
#define  PI               3.14159
#define  C                299792.458      /*  speed of light in km/sec   */
#define  EOF              (-1)            /*  typical end-of-file value  */
#define  MAXINT           2147483647      /*  largest 4-byte integer     */
```

– The use of simple **#define** can improve
  - program clarity
  - program portability

Use of **#define** (2/2)

- Syntactic sugar
  - alter the syntax of C toward users' preference

```
#define  EQ  ==
#define   do              /*   blank   */
```

while (i EQ 1) do { ⇒ **while (i == 1) {**

… **…**

Macros with Arguments (1/5)

- #define can be used to write macro definitions with parameters

  #define *identifier*(*identifier*, …, *identifier*) *token_string*$_{opt}$

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

  SQ(7 + w) ⇒ ((7 + w) * (7 + w))

  SQ(SQ(*p)) ⇒ ((((*p) * (*p))) * (((*p) * (*p))))

  #define SQ(x) x * x

  SQ(a + b) ⇒ a + b * a + b ≠ ((a + b) * (a + b))

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

  4 / SQ(2) ⇒ 4 / (2) * (2) ≠ 4 / ((2) * (2))

Macros with Arguments (2/5)

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

SQ(7) ⇒ (x) ((x) * (x)) (7)

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

if (x == 2)
    x = SQ(y);
else
    ++x;
```

⇒ `x = ((y) * (y));;  /* a common error */`

Macros with Arguments (3/5)

- Macros are frequently used to replace function calls by inline code

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

    m = min(u, v);      ⇒     m = (((u) < (v)) ? (u) : (v));

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

- 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 */
```

Macros with Arguments (4/5)

```
#define    fractional_part       (x - (int) x)

#define    random_char()         (rand() %26 + 'a')
```

- rand()
  - a standard library function defined in stdlib.h
  - returns an integer value randomly distributed between 0 and MAX_RAND

```
#define    random_float()        (rand() % 100 / 10.0)
```

- rand() % 100 : an integer between 0 and 99
- Because 10.0 is of type double, this expression is promoted to a double.
- The value of this expression is a real number of type double between 0 and 9.9

Macros with Arguments (5/5)

```
#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()
```

```
FILL(a, n, “char”);
```

```
if (strcmp(“char”, “char”) == 0)
      for (i=0; i<n; ++i)
          a[i]=random_char();
else
      for (i=0; i<n; ++i)
          a[i] =random_float();
```

- Unlike function definitions, no type checking get done.
- It is the programmer’s responsibility to call the macro with arguments of the appropriate type

Macros in **stdio.h** and **ctype.h** (1/2)

[stdio.h]

```
#define   getchar()    getc(stdin)
#define  putchar(c)   putc(c, stdout)
```

[ctype.h]

Macro	Nonzero (true) is returned if:
isalpha(c)	c is a letter
isupper(c)	c is an uppercase letter
islower(c)	c is an 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

Macros in **stdio.h** and **ctype.h** (2/2)

[ctype.h]

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

- **c** is a variable of integral type, such as **char** or **int**.
- The value of c stored in memory does not get changed

Conditional Compilation

- #if
- #ifdef
- #ifndef
- #endif

- #undef *identifier*

Conditional Compilation

```
#define DEBUG 1
#if DEBUG
    printf(“debug: a=“%d”\n”, a);
#endif
```

```
#define DEBUG
#ifdef DEBUG
    printf(“debug: a=“%d”\n”, a);
#endif
```

```
#include “everything.h”
#undef   PIE
#define  PIE   “I like apple.”
```

```
#if defined(HP9000)||defined(SUN4) && !defined(VAX)
    ….              /* machine-dependent code */
#endif
```

Predefined Macros (ANSI C)

- __DATE__ : a string containing the current date
- __TIME__ : a string containing the current time
- __STDC__ : if the implementation follows ANSI Standard C, the value is a nonzero integer
- __FILE__ : the source file name (string) containing this macro
- __LINE__ : an integer representing the current line number

Assert() Macro (1/2)

```
#define assert(expr)                                    \
    if (!(expr)) {                                      \
        printf("\n%s%s\n%s%s\n%s%d\n\n",    \
                "Assertion failed: ",#expr,             \
                "in file ",__FILE__,                    \
                "at line ",__LINE__);                   \
       abort();                                          \
    }
#endif
```

- unary operator # : "stringization" of formal parameter in a macro definition

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

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

```
int main(void)
{
    printf("Sam" " and " "Debra" ":We love you!\n);
    return 0;
 }
```

Assert() Macro (2/2)

```
        ⋮
        ⋮
assert(n>0 && n<7);
        ⋮
```

```
        ⋮
if (!(n>0 && n<7)) {
    printf("\n%s%s\n%s%s\n%s%d\n\n",
            "Assertion failed: ", "n>0 && n<7",
            "in file ",__FILE__,
            "at line ",__LINE__);
    abort();
};
        ⋮
```