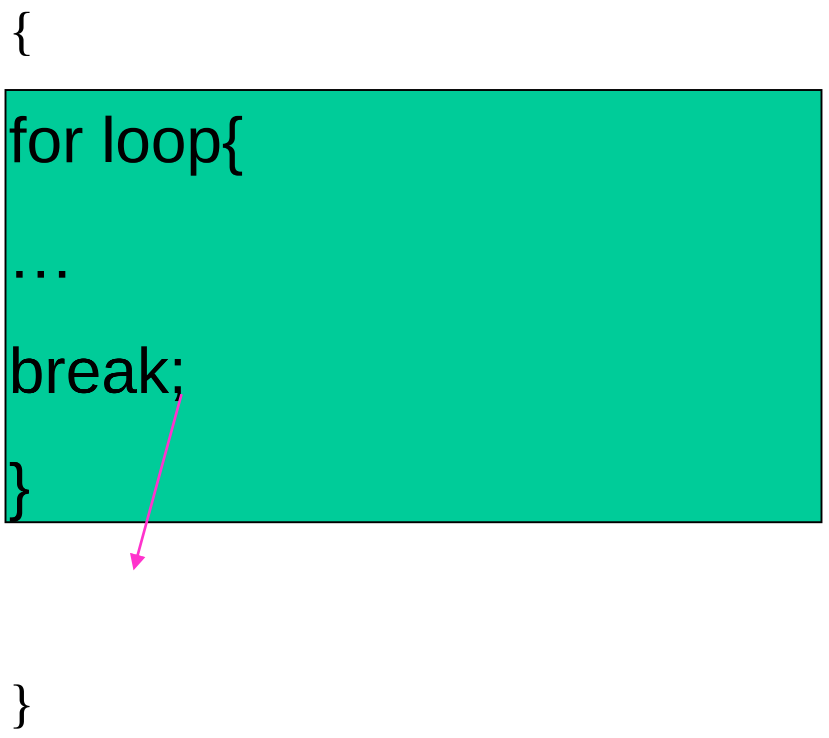
For loop

- **The break and continue Statement**
- **Real World Application: Generating Prime Numbers**
- **The switch Statement**
- **The goto Statement and Labels**
- **The Cast Operator**
- **Real World Application: Summing a Series**
- **The sizeof Operator**
    - **getchar and putchar**
    - **bitwise operator (& | ^ , << >>)**

↓break

- Exits from innermost while, do while, for loop and switch statement



```
{
for loop{
…
break;
}
}
```

```
#include <stdio.h>
main ( ) {
        int I;
        for ( I = 1; I <= 3; I++) {
                printf ( “%d\n”, I );
                if ( I == 2 )
                        break;                  /* exit from for loop */
                printf ( “bottom of loop\n”);
        }
        printf ( “out of loop” );
}
```

The output is

```
1
bottom of loop
2
out of loop
```

Continue

stays within the loop but jumps to the next round

In while loop: jumps to the top and tests expression

In do while loop: jumps to the bottom and tests expression

In for loop: jumps to expression3 and tests expression2

```
#include <stdio.h>
main ( ) {
        int I;
        for ( I = 1; I <= 3; I++) {
                printf ( “%d\n”, I );
                if ( I == 2 )
                        continue; /* jump to execute expression 3 (I++) */
                printf ( “bottom of loop\n”);
        }
        printf ( “out of loop” );
}
```

The output is

```
1
bottom of loop
2
3
bottom of loop
out of loop
```

- Example

  Computes the average of the positive numbers in the standard input.

```
#include <stdio.h>
main ( ) {
    float x, sum = 0.0;
    int count = 0;
    while ( scanf (“%f”, &x ) != EOF ) {
        if ( x <= 0.0 )
            continue; /* skip nonpositive input */
        sum += x;
        count++;
    }
    if ( count > 0 )
        printf ( “\naverage = %f\n”, sum / count );
    else
        printf ( “\nno positive numbers read\n”);
}
```

Exercises 4.1.1

```
for (I=1; I<=6;I++) {
        if( I%2 )
                continue;
        else
                printf(“d\n”,I);
        printf(“last line\n”)
}
```

```
2
last line
4
last line
6
last line
```

Exercise 4.1.2

```
for (I=1; I<=6; I++) {
        if ( I%2)
                printf(“%d\n”,I);
        else
                break;
        printf(“last line\n”)
}
```

1

last line

Real World Application: Generating Prime Numbers

- ***Problem***

  Write a program that prints all positive prime integers less than or equal to n. (A positive integer I is **prime** if I > 1 and the only divisors of I are 1 and I itself.). The value of n is supplied by the user.
- ***Sample Input/Output***

  Input

  12

  Output

  This program lists all primes <= n

  Input n: 12

  Primes <= 12:

  2, 3, 5, 7, 11
- ***Solution***

  1, Get n from standard input;

  2, Use a for loop to step through the integers 2 through n;

  3, Use another for loop to find prime.

- ***C Implementation***

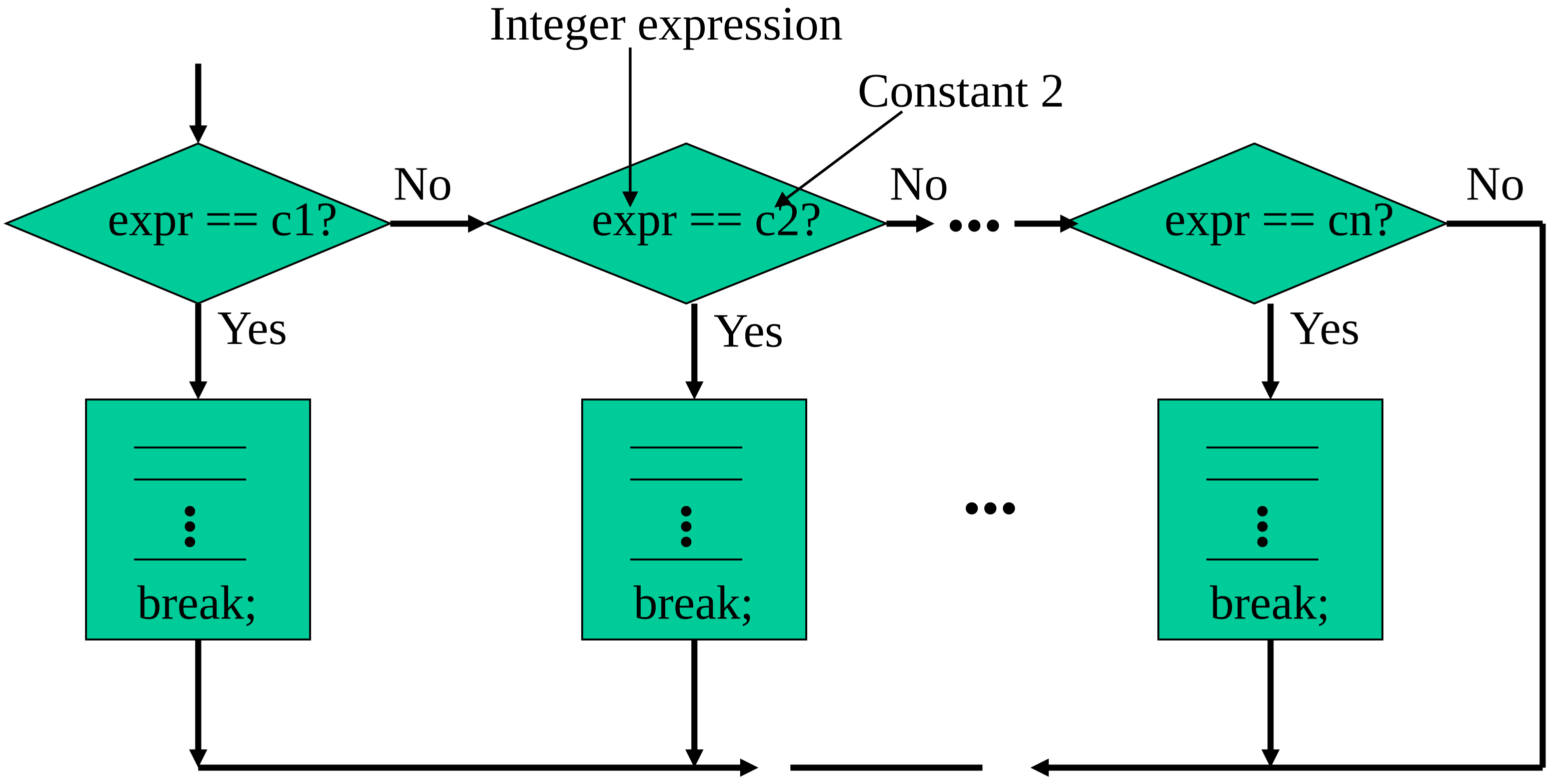
```
#include <stdio.h>
main ( ) {
    int possible_prime, n, possible_divisor;
    printf ( “This program lists all primes <= n\n” );
    printf ( “Input n: ” );
    scanf ( “%d”, &n );
    printf ( “\n\tPrimes <= %d:\n”, n);
    for ( possible_prime = 2; possible_prime <= n; possible_prime++) {
            /* try to find a divisor of possible_prime */
            for ( possible_divisor = 2; possible_divisor < possible_prime;
                    possible_divisor++)
                    if ( possible_prime % possible_divisor == 0 )
                            break;
            if ( possible_divisor == possible_prime )
                    printf ( “%d\n”, possible_prime ); /* exhausted possible divisors,
                                                          so possible_prime is prime */
    }
}
```

↓The switch Statement

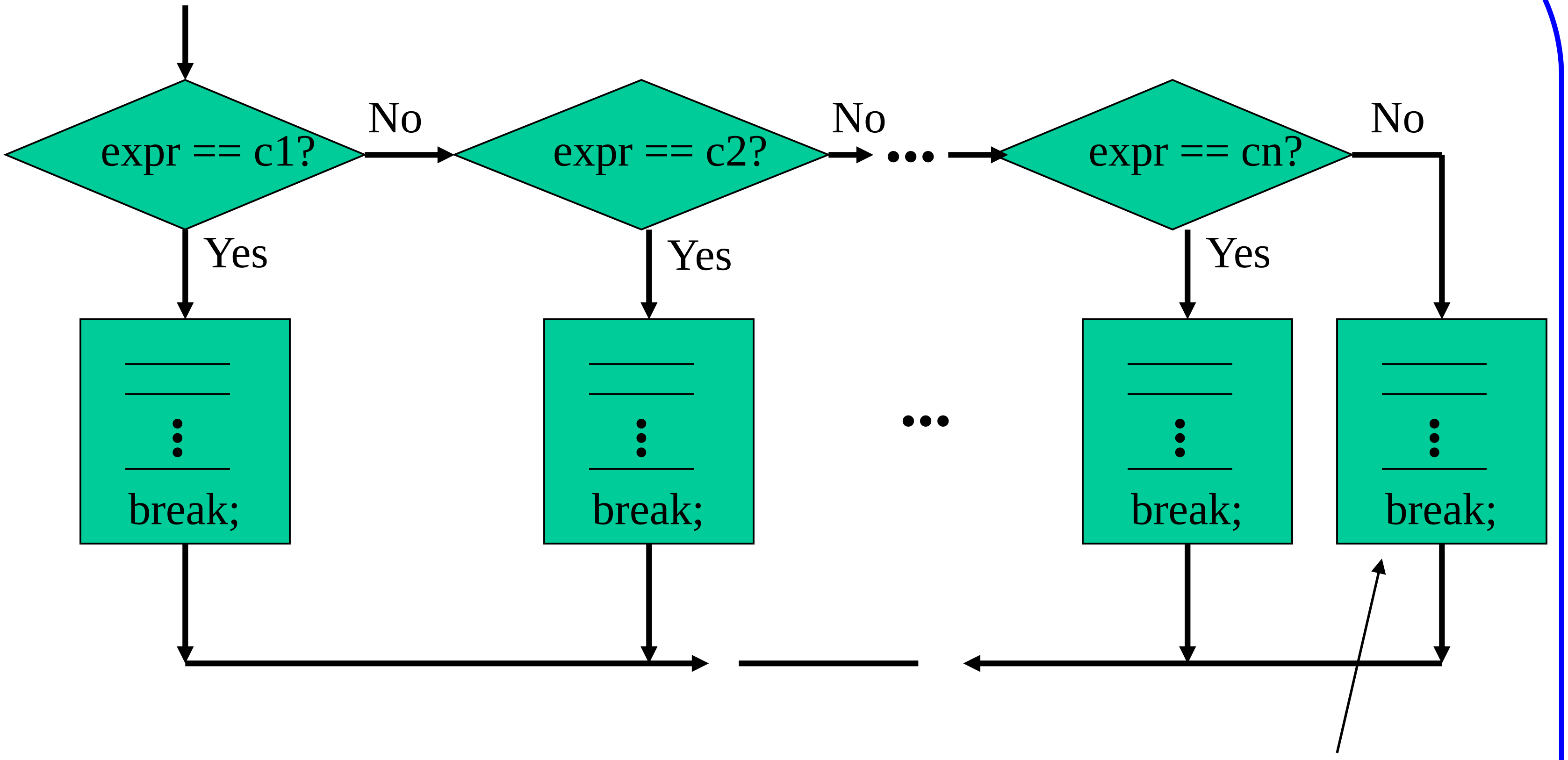
The switch statement can be regarded as a special instance of the if else-if else-if …else-if else statement in which the conditions for branching have integer values.

```
switch ( integer expression ) {
case constant 1:
            statements 1
case constant 2:
            statements 2
case constant 3:
            statements 3
            ...
case constant n:
            statement n
}
```

```
switch ( integer expression ) {
case constant 1:
            statements 1
case constant 2:
            statements 2
            ...
case constant n:
            statement n
default:
            statements
}
```



Flow Chart of Switch Statement



Flow Chart of Switch Statement with Default

```
#include <stdio.h>
main ( ) {              /* switch statement without default case */
        int code;
        code = 4;
        switch (code ) {
                case 1:
                        printf ( “Mechanical Engineering\n” );
                        break;      /* exit from switch */
                case 2:
                        printf ( “Electrical and Computer Engineering\n”);
                        break;
                case 3:
                        printf ( “Science\n” );
                        break;
        }
        printf ( “*** End of course listing ***\n”);
}
```

The output is

```
*** End of course listing ***
```

```
#include <stdio.h>
main ( ) {  /* switch statement with default case */
        int code;
        code = 4;
        switch (code ) {
                case 1:
                        printf ( “Mechanical Engineering\n” );
                        break;              /* exit from switch */
                case 2:
                        printf ( “Electrical and Computer Engineering\n”);
                        break;
                case 3:
                        printf ( “Science\n” );
                        break;
                default:
                        printf ( “No course listed\n” );
                        break;
        }
        printf ( “*** End of course listing ***\n”);
}
```

The output is

```
No course listed
*** End of course listing ***
```

Exercises 4.3.2

```
B=1
switch (B){
case 0:
        printf(“case0\n”);
case 1:
        printf(“case1\n”);
case 2:
        printf(“case2\n”);
}
printf(“end\n”)
```

Exercises 4.3.4

```
B=5
switch (B){
case 0:
      printf(“case0\n”);
case 1:
      printf(“case1\n”);
case 2:
      printf(“case2\n”);
default:
     printf(“default\n);
}
printf(“end\n”)
```

↓The goto Statement and Labels

- The **goto** statement causes an unconditional transfer to some other part of a program.

- A **label** is an identifier followed by a colon:

  identifier:

  A label is a target of a goto statement.
- Caution:

  The goto statement is easily abused. We rarely use the goto statement.

Conditional Expressions

expr 1 ? expr 2 : expr 3 ——— If expr 1 is true, the value of the conditional expression is expr 2; if expr 1 is false, the value of the conditional expression is expr 3.

```
X= q1?  Y : Z;
```

Same as

```
if(q1)
        X=Y;
else
        X=Z;
```

```
for ( I = 1; I < 4; I++ )
        printf ( “%d\n”, ( I % 2 ) ? I : 3 * I );
```

The output is

```
1       /* 1 % 2 = 1, so I %2 is true,  print I ( I = 1 here) */
6       /* 2 % 2 = 0, so I %2 is false, print 3* I (I = 2 here ) */
3       /* 3 % 2 = 1, so I % 2 is true, print I ( I = 3 here ) */
```

↓The Cast Operator

The **cast** operator convert explicitly one data type to another. If x is integer, the value of

```
        ( float ) x
```

is the original value of x converted to float.

Note: the type and value of x are unchanged.

```
#include <stdio.h>
main ( ) {
        int x1 = 3, x2 = 4;
        float y1, y2;
        y1 = x1 / x2;    /* C discards the fractional part of the quotient x1 / x2*/

        y2 = (float) x1 / (float) x2;      /* first convert x1, x2 to float and then to
                                                    compute the quotient x1 / x2*/
        printf ( "y1 = %f, y2 = %f\n", y1, y2 );
}
```

The output is

```
y1 = 0.000000, y2 = 0.750000
```

Real World Application: Summing a Series

- ***Problem***

  Writing a program to sum the first n terms of the infinite series

$$\sum_{i=1}^{\infty}\frac{1}{i^2};$$

- ***Simple Input/Output***

  Input is in color; output is in black.

```
Again ( 1 = yes, 0 = no )? 1
n = ? 1000
```

```
Sum = 1.643934

Again ( 1 = yes, 0 = no )? 1
n = ? 5000
Sum = 1.644734

Again (1 = yes, 0 = no)? 1
n = ? 10000
Sum = 1.644834

Again (1 = yes, 0 = no)? 1
n = ? 20000
Sum = 1.644884

Again (1 = yes, 0 = no)? 0
```

- ***Solution***

- ***C Implementation***

```
#include <stdio.h>
main ( ) {
    int I, n, response;
    float sum;
    do {
            printf ( “\nAgain ( 1 = yes, 0 = no )? “ );
            scanf ( “%d”, &response );
            if ( respone ) {
                    printf ( “\nn = ? ”, &n );
                    scanf ( “%d”, &n);
                    for ( I = 1, sum = 0; I <= n; I++)
                            sum += 1.0 / ( (float) I * (float) I );/* must cast I to float;
                                                    otherwise we would perform integer
                                                    division and obtain zero for I > 1 */
                    printf ( “Sum = %f\n”, sum );
            }
    } while ( response );
}
```

↓The sizeof Operator

C ~~me~~asures storage in bytes.

```
sizeof ( object )
```

is the amount of memory in bytes required to store object.

A program slice,

```
int I;
char c;
sizeof ( I );     /* On PC, the value of sizeof ( I ) is 2 */
sizeof ( c );     /* On any system, the value of sizeof ( c ) is 1 */
```

Note: The values of sizeof (object) may vary from system to system.

↓getchar and putchar

Old way to read and write a single char from standard input

```
scanf ( “%c”, &c);      /* read a single char from standard input  and store in c.*/
printf ( “%c”, c );     /* write a char stored in c to standard output. */
```

New way to read and write a single char from standard input

```
c=getchar( );       /* read a single char from standard input and assign to c*/
putchar( c );       /* write a char stored in the cell of c to standard output. */
```

```
#include <stdio.h>
main ( ) {
        int c;
        while ( (c = getchar ( ) ) != EOF )  /* If you don’t enclose c = getchar() in a
                                pair of braces, getchar() will compare with EOF first */
                putchar ( c );
}
```

In VC++ on Window, Input

```
abcdefg
```

output

```
abcdefg
```

Caution: In above program, the variable c used to handle the characters was defined as type int. The program may not work properly if c is of type char. On some systems, the range of char does not include the value EOF. In this case, the expression

```
( c = getchar ( ) ) != EOF
```

never terminates. C ensures that the range of int includes all character codes in addition to the value EOF.

Notes:

- getchar returns next character from standard input;
- getchar returns EOF if we invoke getchar and there is no character to read;

  --- EOF is different from the integer code of every character used by that system;
- putchar writes the character to the standard output.

  --- Some system writes the output to a buffer, a temporary holding area. The characters in the buffer are not copied to the video display until a carriage return is typed. Above example, the terminal may look like

  ```
  abcdefg
  abcdefg
  ```

  --- Other system write each character to the video display immediately after it is typed. Above example, the terminal may look like

  ```
  aabbccddeeffgg
  ```