Control Statements

What do they do?

- Allow different sets of instructions to be executed depending on the outcome of a logical test.
  - Whether TRUE or FALSE.
  - This is called branching.
- Some applications may also require that a set of instructions be executed repeatedly, possibly again based on some condition.
  - This is called looping.

How do we specify the conditions?

- Using relational operators.
  - Four relation operators: <, <=, >, >=
  - Two equality operations: ==, !=
- Using logical operators / connectives.
  - Two logical connectives: &&, | |
  - Unary negation operator: !

Examples

count <= 100

(math+phys+chem)/3 >= 60

(sex==‘M’) && (age>=21)

(marks>=80) && (marks<90)

(balance>5000) | | (no_of_trans>25)

! (grade==‘A’)

! ((x>20) && (y<16))

The conditions evaluate to …

- Zero
  - Indicates FALSE.
- Non-zero
  - Indicates TRUE.
  - Typically the condition TRUE is represented by the value ‘1’.

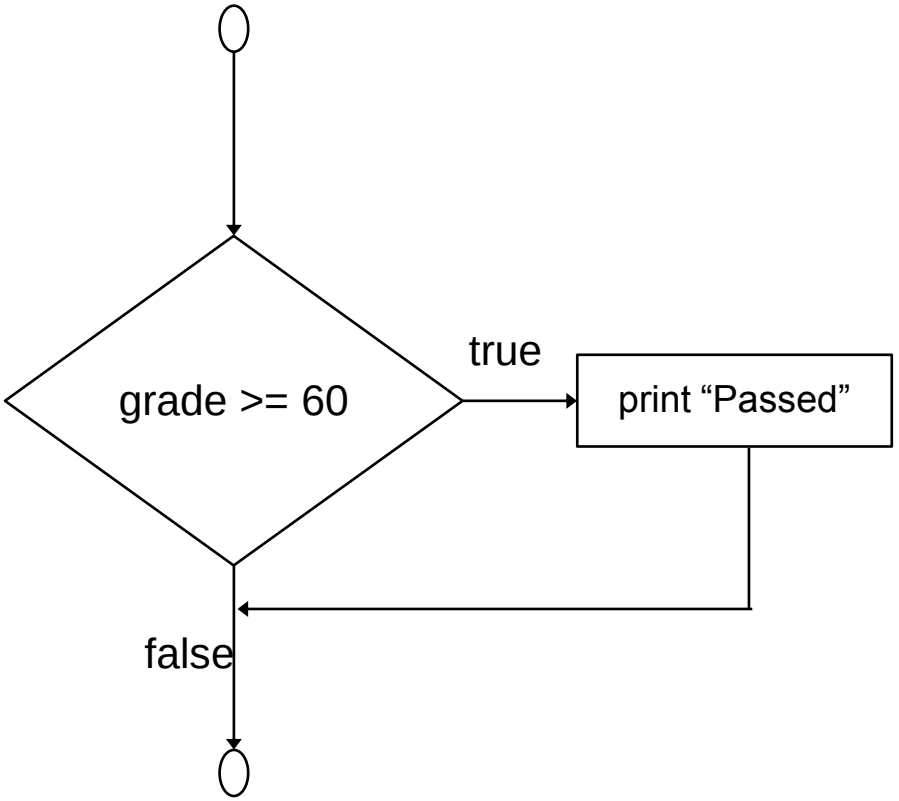
Branching: The if Statement

- Diamond symbol (decision symbol) - indicates decision is to be made.
  - Contains an expression that can be TRUE or FALSE.
  - Test the condition, and follow appropriate path.
- Single-entry / single-exit structure.
- General syntax:

  if (condition) { …….. }

  - If there is a single statement in the block, the braces can be omitted.

The `if` Selection Structure



A decision can be made on any expression.

zero - **`false`**

nonzero - **`true`**

```
if (grade>=60)
  printf(“Passed \n”);
```

Example

```
#include <stdio.h>
main()
{
   int  a,b,c;
   scanf (“%d %d %d”, &a, &b, &c);
   if ((a>=b) && (a>=c))
      printf (“\n The largest number is: %d”, a);
   if ((b>=a) && (b>=c))
      printf (“\n The largest number is: %d”, b);
  if ((c>=a) && (c>=b))
      printf (“\n The largest number is: %d”, c);
}
```

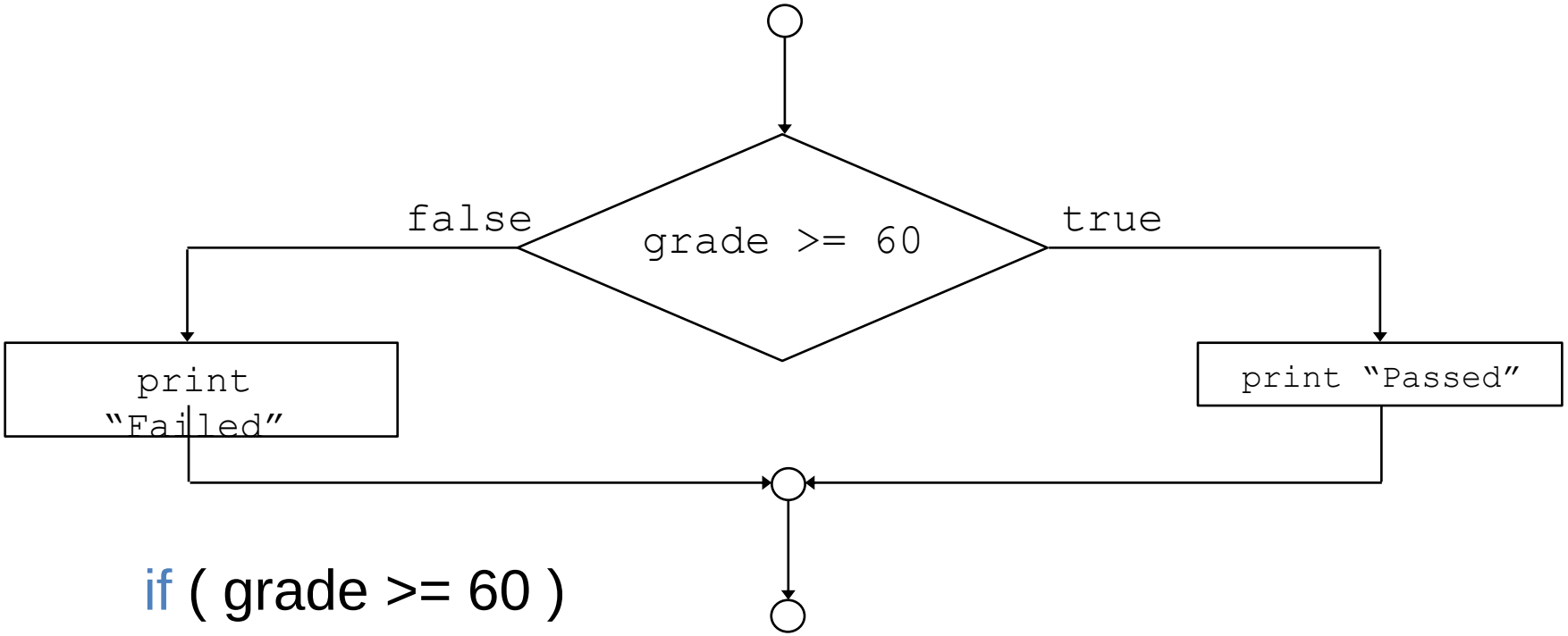
Branching: The if-else Statement

- Also a single-entry / single-exit structure.
- Allows us to specify two alternate blocks of statements, one of which is executed depending on the outcome of the condition.
- General syntax:

```
if  (condition)   { …… block 1 ……. }
else  { …….. block 2 …….. }
```

  - If a block contains a single statement, the braces can be deleted.

The `if/else` Selection Structure




```
if ( grade >= 60 )
      printf( "Passed\n");
else
      printf( "Failed\n");
```

if-else syntax

```
if ( expression )
  {
        statement1;
        statement2;
            .
        statement_n;
   }
```

```
if (grade>=60)
  printf(“Passed \n”);
```

```
if ( expression )
  {
        statement_1;
        statement_2;
            .
        statement_n;
   }
else
{
    Statement_1;
        .
    Statement_m;
}
```

```
if ( grade >= 60 )
      printf( "Passed\n");
else
      printf( "Failed\n");
```

Nesting of if-else Structures

- It is possible to nest if-else statements, one within another.
- All if statements may not be having the “else” part.
  - Confusion??
- Rule to be remembered:
  - An “else” clause is associated with the closest preceding unmatched “if”.

```
if e1 s1
else if e2 s2
```

```
if e1 s1
else if e2 s2
else s3
```

?

```
if e1 if e2 s1
else s2
else s3
```

```
if e1 if e2 s1
else s2
```

```
if e1 s1
else if e2 s2
```

```
if e1 s1
else if e2 s2
else s3
```

```
if e1 if e2 s1
else s2
else s3
```

```
if e1 if e2 s1
else s2
```

```
if  e1  s1
        else  if  e2  s2
```

```
if  e1  s1
else  if  e2  s2
      else s3
```

```
 if  e1  if  e2  s1
        else  s2
else  s3
```

```
if  e1  if  e2  s1
        else s2
```

Example

```
#include <stdio.h>
main()
{
   int  a,b,c;
   scanf (“%d %d %d”, &a, &b, &c);
   if (a>=b)
        if (a>=c)
       printf (“\n The largest number is: %d”, a);
        else       printf (“\n The largest number is: %d”, c);
  else
        if (b>=c)
       printf (“\n The largest number is: %d”, b);
        else       printf (“\n The largest number is: %d”, c);
}
```

Example

```
#include <stdio.h>
main()
{
   int  a,b,c;
   scanf (“%d %d %d”, &a, &b, &c);
   if ((a>=b) && (a>=c))
      printf (“\n The largest number is: %d”, a);
   else if (b>c)
      printf (“\n The largest number is: %d”, b);
  else
      printf (“\n The largest number is: %d”, c);
}
```

Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
  - Does not ordinarily cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are `true`, zero values are `false`
- Example:

```
if ( payCode == 4 )
   printf( "You get a bonus!\n" );
```

Checks paycode, if it is 4 then a bonus is awarded

Equality check improper

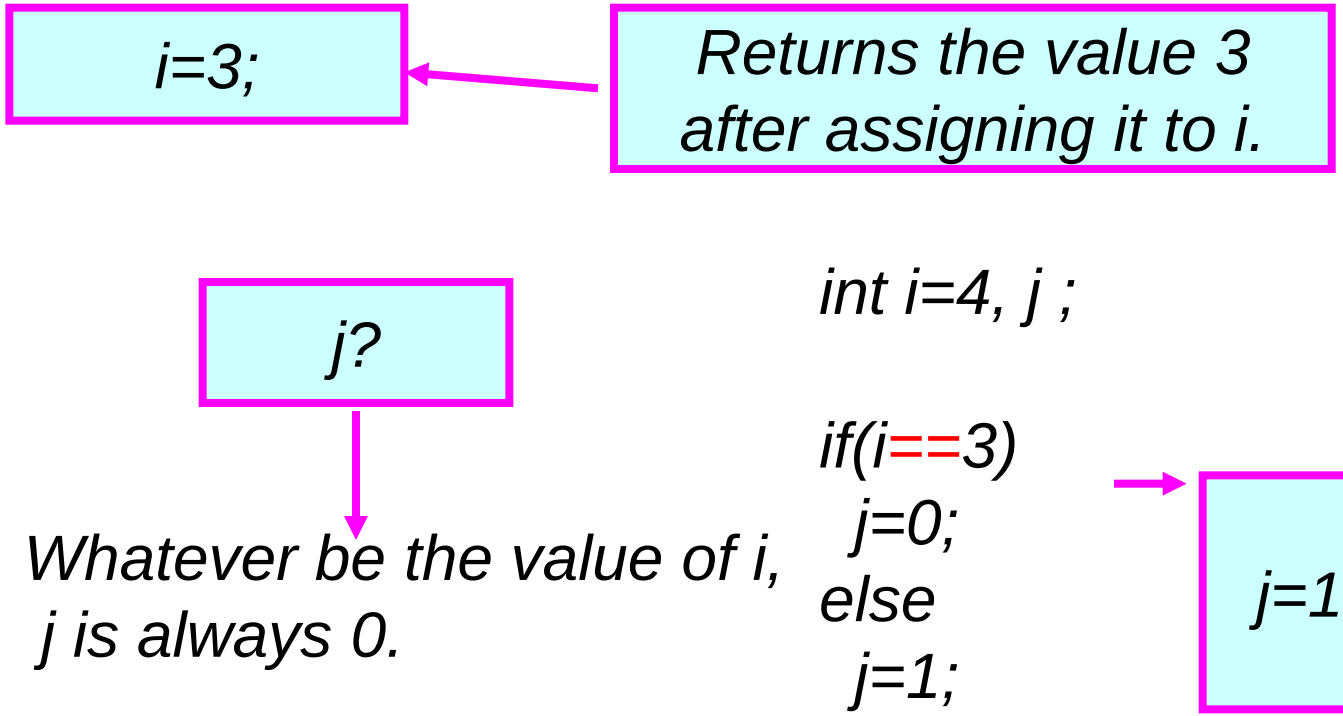
```
if ( payCode = 4 )
   printf( "You get a bonus!\n" );
```

Equality check proper

```
if ( payCode == 4 )
   printf( "You get a bonus!\n" );
```

Generalization of expression evaluation in C

- Assignment (=) operation is also a part of expression.



More about expressions

- Increment (++) and Decrement (--)Operations

Prefix operation: *++i;* *--i;*

First increment / decrement and then used in evaluation

Postfix operation: *i++;* *i--;*

increment / decrement operation after being used in evaluation

```
int t,m=1;
t=++m;
```

→ *m=2; t=2;*

```
int t,m=1;
t=m++;
```

→ *m=2; t=1;*

Some More Examples

Initial values :: a = 10; b = 20;

x = 50 + ++a;

a = 11, x = 61

x = 50 + a++;

x = 60, a = 11

x = a++ + --b;

b = 19, x = 29, a = 11

x = a++ – ++a;

Undefined value (implementation dependent)

Ternary conditional operator (? :)

- Takes three arguments (condition, value if `true`, value if `false`)
- `Returns the evaluated value accordingly.`

grade >= 60 ? printf(“Passed\n”) : printf(“Failed\n”);

(expr1)? (expr2): (expr3);

Example:

interest = (balance>5000) ? balance*0.2 : balance*0.1;

Returns a value

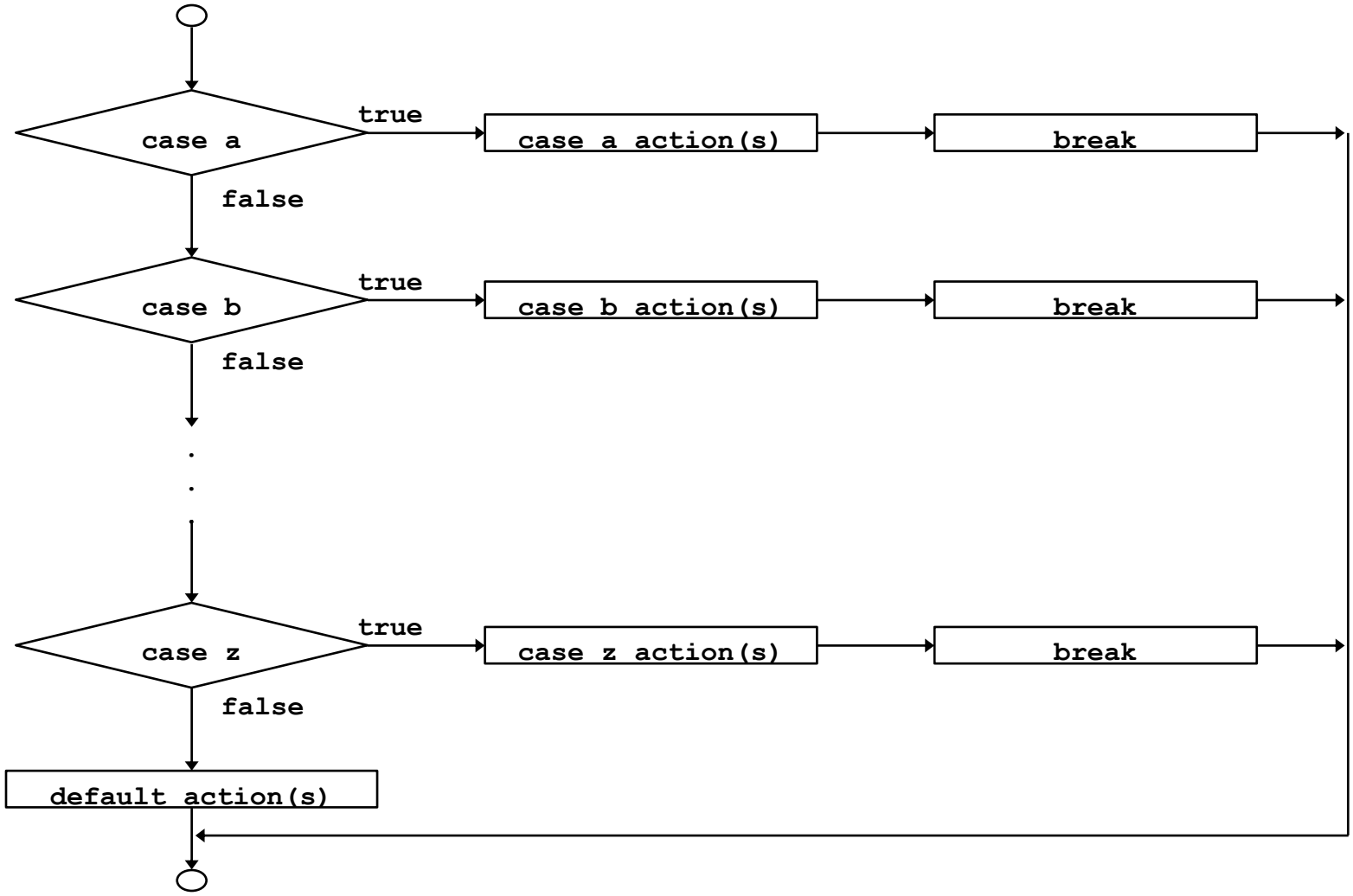
The switch Statement

- This causes a particular group of statements to be chosen from several available groups.
  - Uses “switch” statement and “case” labels.
  - Syntax of the “switch” statement:

```
switch (expression)  {
   case expression-1: { …….. }
   case expression-2: { …….. }

   case expression-m: { …….. }
   default: { …….. }
}
```

The `switch` Multiple-Selection Structure



Example

```
switch ( letter ) {
    case 'A':
        printf("First letter\n");
        break;
    case 'Z':
        printf("Last letter\n");
        break;
    default :
        printf("Middle letter\n");
        break;
}
```

Example

```
switch  (choice = toupper(getchar()))  {

    case ‘R’:       printf (“RED \n”);
                    break;
    case ‘G’:       printf (“GREEN \n”);
                    break;
    case ‘B’:       printf (“BLUE \n”);
                    break;
    default:        printf (“Invalid choice \n”);

}
```

Example

```
switch  (choice = getchar())  {

    case ‘r’:
    case ‘R’:       printf (“RED \n”);
                    break;
    case ‘g’:
    case ‘G’:       printf (“GREEN \n”);
                    break;
    case ‘b’:
    case ‘B’:       printf (“BLUE \n”);
                    break;
    default:        printf (“Invalid choice \n”);

}
```

The break Statement

- Used to exit from a switch or terminate from a loop.
  - Already illustrated in the switch examples.
- With respect to “switch”, the “break” statement causes a transfer of control out of the entire “switch” statement, to the first statement following the “switch” statement.

The Essentials of Repetition

- Loop
  - Group of instructions computer executes repeatedly while some condition remains true
- Counter-controlled repetition
  - Definite repetition - know how many times loop will execute
  - Control variable used to count repetitions

- Sentinel-controlled repetition
  - Indefinite repetition
  - Used when number of repetitions not known
  - Sentinel value indicates "end of data"

Counter-Controlled Repetition

- Counter-controlled repetition requires
  - *name* of a control variable (or loop counter).
  - *initial value* of the control variable.
  - condition that tests for the *final value* of the control variable (i.e., whether looping should continue).
  - *increment* (or *decrement*) by which the control variable is modified each time through the loop.

```
int counter =1;                 //initialization
while (counter <= 10) {         //repetition condition
                        printf( "%d\n", counter );
                        ++counter;              //increment
                      }
```

Counter-Controlled Repetition

- Counter-controlled repetition requires
  - *name* of a control variable (or loop counter).
  - *initial value* of the control variable.
  - condition that tests for the *final value* of the control variable (i.e., whether looping should continue).
  - *increment* (or *decrement*) by which the control variable is modified each time through the loop.

```
int counter;
for (counter=1;counter<=10;counter++)
    printf("%d\n",counter);
```

while Statement

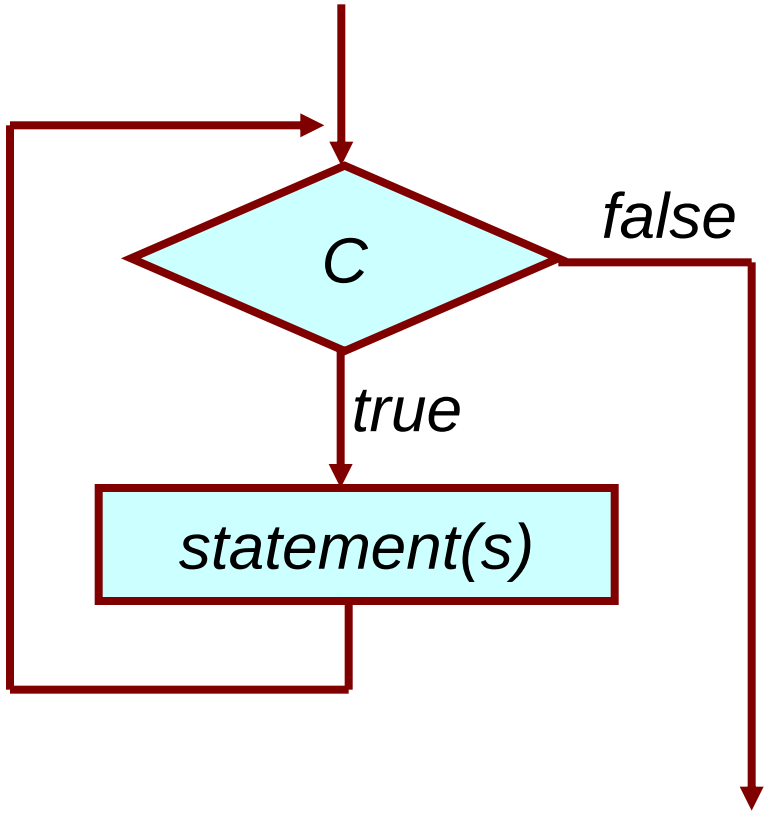
```
while (condition)
    statement_to_repeat;

while (condition) {
        statement_1;
        ...
        statement_N;
}
```

```
int  digit = 0;

while  (digit <= 9)
  printf ("%d \n", digit++);
```

```
/* Weight loss program */
while ( weight > 65 ) {
    printf("Go, exercise, ");
    printf("then come back. \n");
    printf("Enter your weight: ");
    scanf("%d", &weight);
    }
```



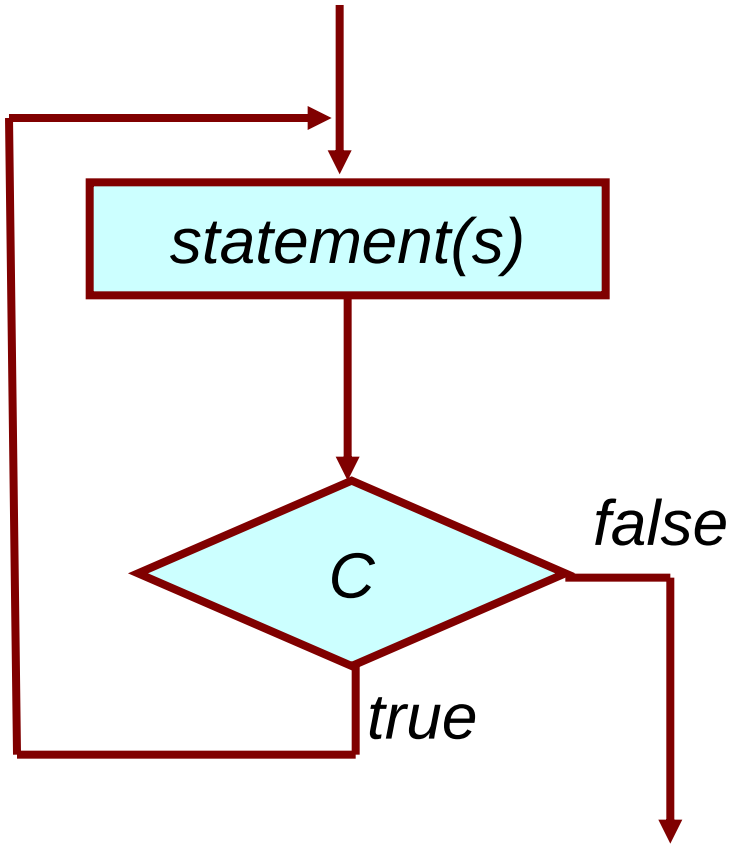
Single-entry / single-exit structure

do-while Statement

```
do {
    statement-1
    statement-2
        .
        .
    statement-n
} while ( condition );
```

```
/* Weight loss program */
do {
    printf("Go, exercise, ");
    printf("then come back. \n");
    printf("Enter your weight: ");
    scanf("%d", &weight);
  } while ( weight > 65 ) ;
```

At least one round of exercise ensured.



Single-entry / single-exit structure

```
int  digit = 0;

do
    printf ("%d \n", digit++);
while (digit <= 9);
```

for Statement

```
for (initial; condition; iteration)
        statement_to_repeat;
```

```
for (initial; condition; iteration) {
        statement_1;
        ...
        statement_N;
}
```

All are expressions.
initial → expr1
condition → expr2
iteration →expr3

```
fact = 1;  /* Calculate 10 ! */
for ( i = 1; i < =10; i++)
    fact = fact * i;
```

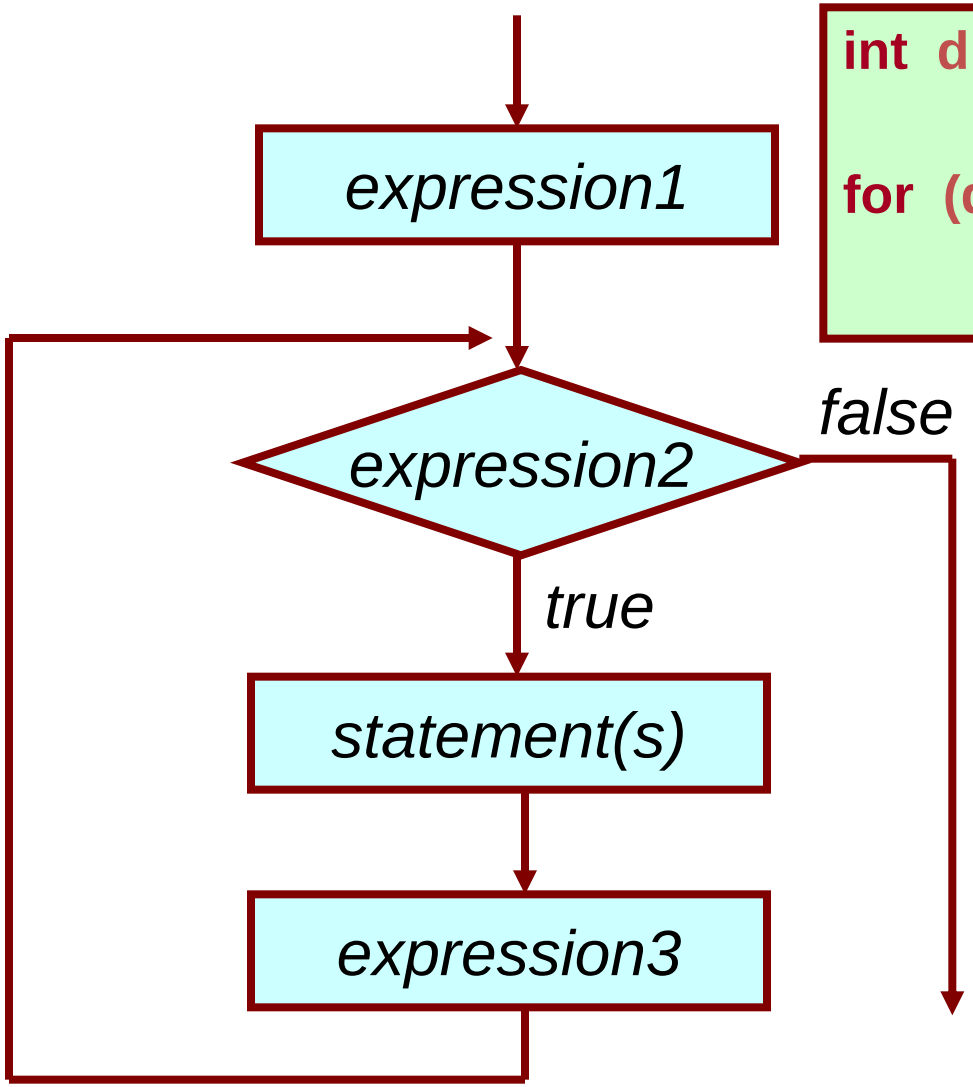
No semicolon after last expression

- How it works?
  - "expression1" is used to *initialize* some variable (called *index*) that controls the looping action.
  - "expression2" represents a *condition* that must be true for the loop to continue.
  - "expression3" is used to *alter* the value of the *index* initially assigned by "expression1".

```
int  digit;

for  (digit=0; digit<=9; digit++)
            printf ("%d \n", digit);
```

- How it works?
  - "expression1" is used to *initialize* some variable (called *index*) that controls the looping action.
  - "expression2" represents a *condition* that must be true for the loop to continue.
  - "expression3" is used to *alter* the value of the *index* initially assigned by "expression1".




```
int  digit;

for  (digit=0; digit<=9; digit++)
        printf ("%d \n", digit);
```

Single-entry / single-exit structure

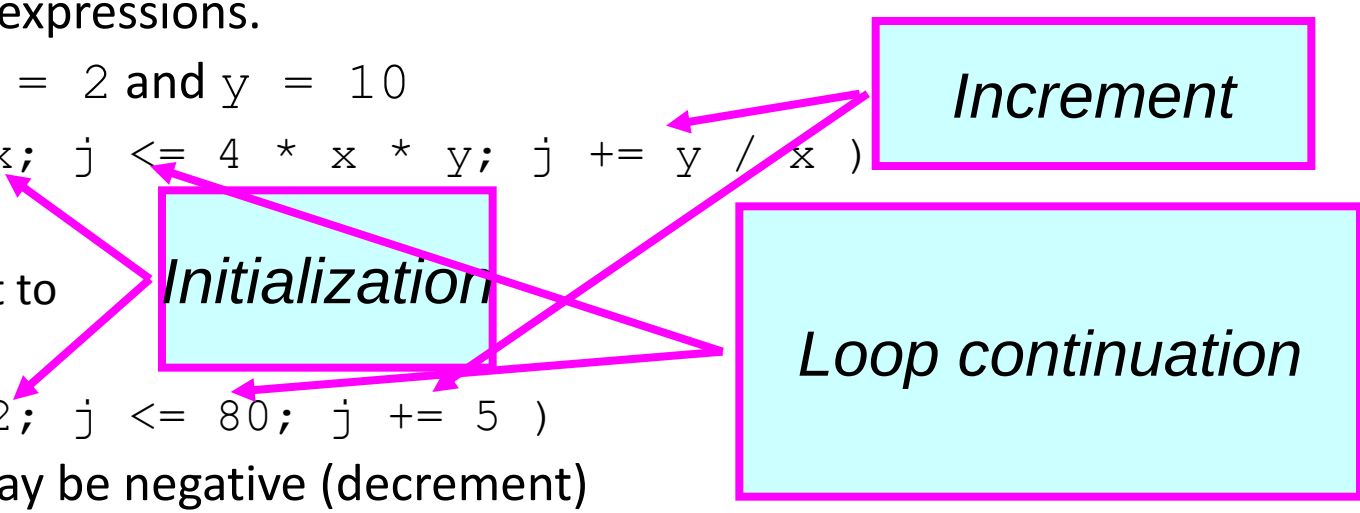
The For Structure: Notes and Observations

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions.
  - e.g. Let `x = 2` and `y = 10`

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to

```
for ( j = 2; j <= 80; j += 5 )
```



- "Increment" may be negative (decrement)
- If loop continuation condition initially `false`
  - Body of `for` structure not performed
  - Control proceeds with statement after `for` structure

for :: Examples

```
int  fact = 1, i;

for  (i=1; i<=10; i++)
     fact = fact * i;
```

```
int  sum = 0, N, count;

scanf ("%d", &N);

for (i=1; i<=N, i++)
     sum = sum + i * i;

printf ("%d \n", sum);
```

- The comma operator
  - We can give several statements separated by commas in place of “expression1”, “expression2”, and “expression3”.

```
for  (fact=1, i=1; i<=10; i++)
   fact = fact * i;

for (sum=0, i=1; i<=N, i++)
   sum = sum + i * i;
```

Specifying “Infinite Loop”

```
while  (1)  {
   statements
}
```

```
for  (; ;)
{
     statements
}
```

```
do  {
   statements
}  while (1);
```

break Statement

- Break out of the loop { }
  - can use with
    - *while*
    - *do while*
    - *for*
    - *switch*
  - does not work with
    - *if {}*
    - *else {}*

Causes immediate exit from a while**,** for**,** do**/**while **or** switch **structure**

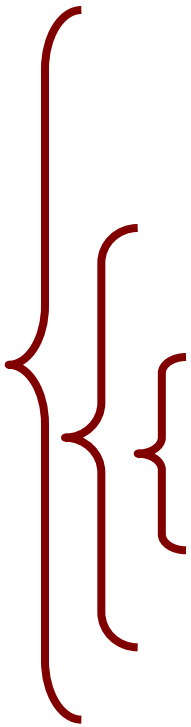
Program execution continues with the first statement after the structure

Common uses of the break **statement**

Escape early from a loop

Skip the remainder of a switch **structure**

A Complete Example



```
#include  <stdio.h>
main()
{
     int  fact, i;

     fact = 1;  i = 1;

     while  ( i<10 )   {                  /* run loop –break when fact >100*/
               fact = fact * i;
               if ( fact > 100 )  {
                         printf ("Factorial of %d  above 100", i);
                         break;                  /* break out of the while loop */
               }
               i ++ ;
     }
}
```

`continue` Statement

- `continue`
  - Skips the remaining statements in the body of a `while`, `for` or `do/while` structure
    - Proceeds with the next iteration of the loop
  - `while` and `do/while`
    - Loop-continuation test is evaluated immediately after the `continue` statement is executed
  - `for` structure
    - Increment expression is executed, then the loop-continuation test is evaluated.

    expression3 is evaluated, then *expression2* is evaluated.

An Example with “break” & “continue”

```
fact = 1; i = 1;                    /* a program to calculate 10 !
while  (1)  {
     fact = fact * i;
     i ++ ;
     if  ( i<10 )
              continue;             /* not done yet ! Go to loop and
                                    perform next iteration*/
     break;
}
```

Some Examples

Example 1: Test if a number is prime or not

```
#include <stdio.h>
main()
{
    int  n, i=2;
    scanf (“%d”, &n);
    while (i < n)  {
        if (n % i == 0)  {
            printf (“%d is not a prime \n”, n);
            exit;
        }
        i++;
    }
    printf (“%d is a prime \n”, n);
}
```

More efficient??

```
#include <stdio.h>
main()
{
    int  n, i=3;
    scanf (“%d”, &n);
    while (i < sqrt(n))  {
        if (n % i == 0)  {
            printf (“%d is not a prime \n”, n);
            exit;
        }
        i = i + 2;
    }
    printf (“%d is a prime \n”, n);
}
```

Example 2: Find the sum of digits of a number

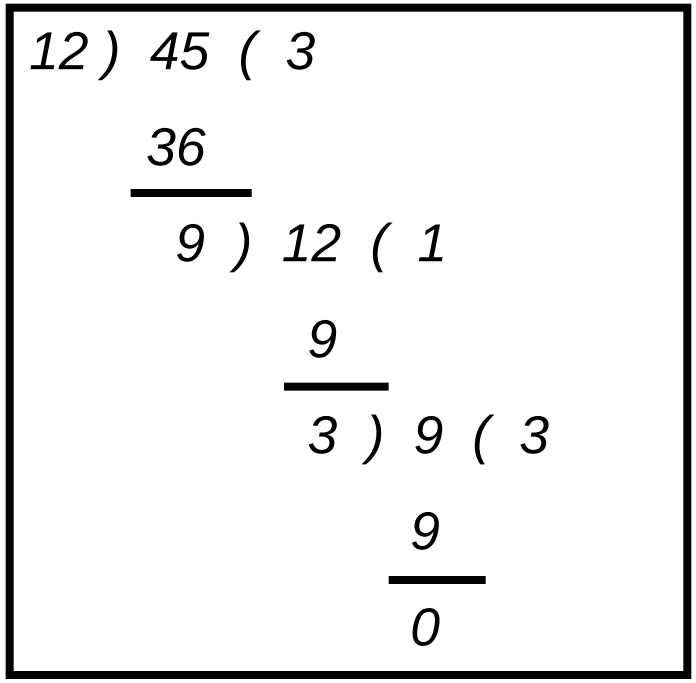
```
#include  <stdio.h>
main()
{
    int     n, sum=0;
    scanf (“%d”, &n);
    while (n != 0)  {
            sum = sum + (n % 10);
            n = n / 10;
    }
    printf (“The sum of digits of the number is %d \n”, sum);
}
```

Example 3: Decimal to binary conversion

```
#include  <stdio.h>
main()
{
     int  dec;
     scanf (“%d”, &dec);
     do
     {
          printf (“%2d”,  (dec % 2));
          dec = dec / 2;
     }  while (dec != 0);
     printf (“\n”);
}
```

Example 4: Compute GCD of two numbers

```
#include  <stdio.h>
main()
{
     int  A, B, temp;
     scanf (%d %d", &A, &B);
     if  (A > B)  { temp = A;  A = B;  B = temp; }
     while ((B % A) != 0)  {
               temp = B % A;
               B = A;
               A = temp;
     }
     printf ("The GCD is %d", A);
}
```



Initial: A=12, B=45
Iteration 1: temp=9, B=12,A=9
Iteration 2: temp=3, B=9, A=3
B % A = 0 ➔ GCD is 3

Shortcuts in Assignments

- Additional assignment operators:

  + =, – =, * =, / =, % =

  a += b is equivalent to a = a + b

  a *= (b+10) is equivalent to a = a * (b + 10)

  and so on.

More about scanf and printf

Entering input data :: scanf function

- General syntax:

  scanf (control string, arg1, arg2, …, argn);

  – “control string refers to a string typically containing data types of the arguments to be read in;

  – the arguments arg1, arg2, … represent pointers to data items in memory.

  Example: scanf (%d %f %c”, &a, &average, &type);

- The control string consists of individual groups of characters, with one character group for each input data item.

  – ‘%’ sign, followed by a conversion character.

– Commonly used conversion characters:

c	single character
d	decimal integer
f	floating-point number
s	string terminated by null character
X	hexadecimal integer

– We can also specify the maximum field-width of a data item, by specifying a number indicating the field width before the conversion character.

Example: scanf (“%3d %5d”, &a, &b);

Writing output data :: printf function

- General syntax:

  printf (control string, arg1, arg2, …, argn);

  – “control string refers to a string containing formatting information and data types of the arguments to be output;

  – the arguments arg1, arg2, … represent the individual output data items.

- The conversion characters are the same as in scanf.

- Examples:

```
printf  (“The average of %d and %d is %f”, a, b, avg);
printf  (“Hello \nGood \nMorning \n”);
printf (“%3d %3d %5d”, a, b, a*b+2);
printf (“%7.2f  %5.1f”, x, y);
```

- Many more options are available:
  - Read from the book.
  - Practice them in the lab.
- String I/O:
  - Will be covered later in the class.