

Structured Programming

Dr. Mohamed Khedr

Lecture 6

<http://webmail.aast.edu/~khedr>

Program Control

- Standard C Statements

Outline

- This Topic Introduces
 - selection structure
 - **if**
 - **if/else**
 - repetition control structures
 - **While**
 - additional repetition control structures
 - **for**
 - **do/while**
 - **switch** additional multiple selection structure
 - **break** statement
 - Used for exiting immediately and rapidly from certain control structures
 - **continue** statement
 - Used for skipping the remainder of the body of a repetition structure and proceeding with the next iteration of the loop

Control Structures

- **C allows a program to make a decision based on the value of a condition. Such a condition must evaluate to true or false.**

use logical expressions which may include:

6 Relational Operators

< <= > >= == !=

3 Logical Operators

! && ||

LOGICAL

EXPRESSION

MEANING

DESCRIPTION

! p	NOT p	! p is false if p is true ! p is true if p is false
p && q	p AND q	p && q is true if both p and q are true. It is false otherwise.
p q	p OR q	p q is true if either p or q or both are true. It is false otherwise.

Equality Vs Assignment

- **Given:**
if (grade = 100)
 printf (“Perfect Score!”);
- **This statement does the following:**
 - **Grade is assigned the value 100.**
 - **Because 100 is true (i.e. non-zero!), the condition is always true.**
- **No matter what the student grade, it always says “Perfect Score!”**

Use of blocks recommended

```
if ( Expression )
```

```
{
```

```
}
```

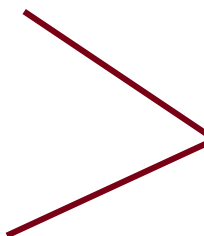
```
else
```

```
{
```

```
}
```



“if clause”



“else clause”

```
int  carDoors, driverAge ;
double premium, monthlyPayment ;
. . .
if ( (carDoors == 4 ) && (driverAge > 24) )
{
    premium = 650.00 ;
    printf( " LOW RISK " ) ;
}
else
{
    premium = 1200.00 ;
    printf("HIGH RISK ") ;
}

monthlyPayment = premium / 12.0 + 5.00 ;
```


What happens if you omit braces?

```
if ( (carDoors == 4 ) && (driverAge > 24) )
    premium = 650.00 ;
    printf( " LOW RISK " ) ;
else
    premium = 1200.00 ;
    printf( " HIGH RISK " ) ;

monthlyPayment = premium / 12.0 + 5.00 ;
```

COMPILE ERROR OCCURS. The "if clause" is the single statement following the if.

Braces can only be omitted when
each clause is a single statement

```
if ( lastInitial <= 'K' )  
    volume = 1;  
else  
    volume = 2;  
  
printf( "Look it up in volume # %d of the phone  
book", volume ) ;
```

What output? and Why?

```
int code;  
code = 0;  
if ( ! code )  
    printf( “Yesterday”);  
else  
    printf( “Tomorrow”);
```

What output? and Why?

```
int age;  
age = 30;  
if ( age < 18 )  
    printf( “Do you drive?”);  
    printf( “Too young to vote”);
```

What output? and Why?

```
int age;  
  
age = 20;  
  
if ( age == 16 )  
{  
    printf( "Did you get driver's license?" ) ;  
}
```

If--Else for a mail order

Write a program to calculate the total price of a certain purchase. There is a discount and shipping cost:

- **The discount rate is 25% and the shipping is 10.00 if purchase is over 100.00.**
- **Otherwise, The discount rate is 15% and the shipping is 5.00 pounds.**

These braces cannot be omitted

```
if ( purchase > 100.00 )
{
    discountRate = .25 ;
    shipCost = 10.00 ;
}
else
{
    discountRate = .15 ;
    shipCost = 5.00 ;
}

totalBill = purchase * (1.0 - discountRate) + shipCost ;
```

Example

- Write a program to ask a student for his grades in 3 exams (each out of 50) , get their total and inform the student whether he passed or failed the course.

Example

The Air Force has asked you to write a program to label aircrafts as military or civilian. Your program input is the plane's speed and its estimated length. For planes traveling faster than 1100 km/hr, you will label those shorter than 52 m "military", and longer as "Civilian". For planes traveling less than 1100, you will issue an "aircraft unknown" statement.

if else statement

```
// multiple selection
if ( expression1 )
{
    statements1    // expression1 is true
}
else if ( expression2 )
{
    statements2    // expression2 is true
}
else
{
    statements3    // none are true
}

statements4
```

Multi-way Branching

```
if ( creditsEarned >= 90 )  
    printf ( "Fourth year student " ) ;  
else if ( creditsEarned >= 60 )  
    printf ( "Third year student " ) ;  
else if ( creditsEarned >= 30 )  
    printf ( "Second year student " ) ;  
else  
    printf ( "First year student " ) ;
```

// program tells how to spend your day

```
#include < stdio.h >  
  
void main ( )  
{    int    day;  
      char  raining;  
  
      printf( “Enter day (use 1 for Sunday)” );  
      scanf( “%d”, &day );  
  
      printf( “Is it raining? (Y/N)” );  
      scanf( “%c”, &raining );
```

```
if ( ( day == 6) || (day == 7) )    /* Fri or Sat */
    {
        if (raining == 'Y')
            printf( "Read in bed");
        else
            printf( "Have fun outdoors");
    }
else
    {
        printf( "Go to class ");
        if (raining == 'Y')
            printf( "Take an umbrella");
    }
} /* End of Program */
```

Switch statement

- Used to select one of several alternatives
- BASED on the value of a single variable.
- This variable may be an int or a char but **NOT** a float (or double).

switch statement

```
// multiple selection
switch ( integral expression )
{
  case constant integral expression1:
    statements1      // expression1 matches
    break;
  case constant integral expression2:
    statements2      // expression2 matches
    break;
  case constant integral expression3:
    statements3      // expression3 matches
    break;
  default:              // no expression matches
    statements4
    break;
}
```

Example

```
char grade ;
printf("Enter your letter grade: ");
scanf("%c", &grade);
switch ( grade )
{
    case 'A' : printf(" Excellent Job");
                break;
    case 'B' : printf ( " Very Good ");
                break;
    case 'C' : printf(" Not bad ");
                break;
    case 'F' : printf("Faiing");
                break;
    default : printf(" Wrong Input ");
}
}
```


Light bulbs

Write a program to ask the user for the brightness of a light bulb (in Watts), and print out the expected lifetime:

<u>Brightness</u>	<u>Lifetime in hours</u>
25	2500
40, 60	1000
75, 100	750
otherwise	0

```
int bright ;
printf("Enter the bulb brightness: ");
scanf("%d", &bright);
switch ( bright )
{
    case 25 : printf(" Expected Lifetime is 2500 hours");
              break;
    case 40 :
    case 60 : printf ( "Expected Lifetime is 1000 hours ");
              break;
    case 75 :
    case 100 : printf("Expected Lifetime is 750 hours ");
              break;
    default : printf("Wrong Input ");
}
}
```

break vs return

- **break** means exit the switch statement and continue on with the rest of the program.
- **return** means exit the whole program.
- They could both be used anywhere in the program.

Testing Selection Control Structures

- to test a program with branches, use enough data sets so that every branch is executed at least once
- this is called **minimum complete coverage**

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"

Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires

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

Example:

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

- The statement

```
int counter = 1;
```

- Names **counter**
- Declares it to be an integer
- Reserves space for it in memory
- Sets it to an initial value of **1**
- This is **not** an executable statement, it is a declaration.

Repetition Structure: while

```
1  /* Fig. 3.6: fig03_06.c
2  Class average program with
3  counter-controlled repetition */
4  #include <stdio.h>
5
6  int main()
7  {
8  int counter, grade, total, average;
9
10 /* initialization phase */
11 total = 0;
12 counter = 1;
13
14 /* processing phase */
15 while ( counter <= 10 ) {
16     printf( "Enter grade: " );
17     scanf( "%d", &grade );
18     total = total + grade;
19     counter = counter + 1;
20 }
21
22 /* termination phase */
24 printf( "Class average is %d\n", average );
25
26 return 0; /* indicate program ended successfully */
27 }
```

```
printf( "Enter grade, -1 to end: " );
scanf( "%d", &grade );
while ( grade != -1 ) {
    total = total + grade;
    counter = counter + 1;
    printf( "Enter grade, -1 to end: " );
    scanf( "%d", &grade );
} /* termination phase */
if ( counter != 0 ) {
    average = ( float ) total / counter;
    printf( "Class average is %.2f", average );
}
else
    printf( "No grades were entered\n" );
```

Program Output:

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

Repetition Structure: **for**

- **for** loops syntax


```
for ( initialization ; loopContinuationTest ; increment )  
    statement
```

Example: Prints the integers from one to ten

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

- For loops can usually be rewritten as **while** loops:

```
initialization;  
while ( loopContinuationTest ) {  
    statement;  
    increment;  
}
```



No semicolon
(;) after last
expression

- Initialization and increment

- Can be comma-separated list of statements

Example:

```
for ( i = 0, j = 0; j + i <= 10; j++, i++)  
    printf( "%d\n", j + i );
```


The `for` Structure (cont.)

- Arithmetic expressions

- Initialization, loop-continuation, and increment can contain arithmetic expressions. If `x` equals `2` and `y` equals `10`

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

is equivalent to

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

- Notes about the `for` structure:

- "Increment" may be negative (decrement)
- If the loop continuation condition is initially **false**
 - The body of the `for` structure is not performed (i.e. pre-test)
 - Control proceeds with the next statement after the `for` structure
- Control variable
 - Often printed or used inside for body, but not necessarily

The for Structure (cont.)

```
1  /* Fig. 4.5: fig04_05.c
2     Summation with for */
3  #include <stdio.h>
4
5  int main()
6  {
7     int sum = 0, number;
8
9     for ( number = 2; number <= 100; number += 2 )
10        sum += number;
11
12    printf( "Sum is %d\n", sum );
13
14    return 0;
15 }
```

1. Initialize variables

2. for repetition structure

Program Output:

Sum is 2550

$2 + 4 + 8 + \dots + 100 = 2550$

Repetition Structure: do/while

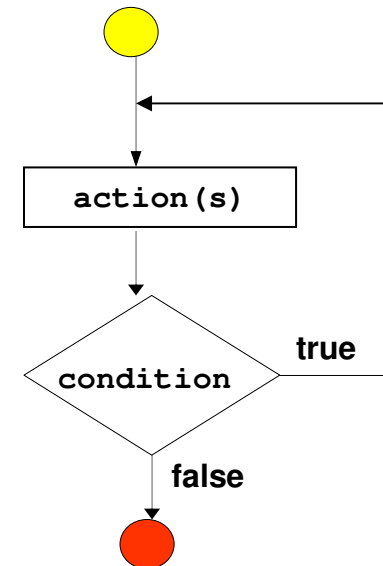
- The **do/while** repetition structure
 - Similar to the **while** structure
 - do/while is a “post-test” condition. The body of the loop is performed at least once.
 - All actions are performed at least once
 - Format:

```
do {  
    statement;  
} while ( condition );
```

Example: Prints the integers from 1 to 10.
(letting counter = 1):

```
do {  
    printf( "%d  ", counter );  
} while (++counter <= 10);
```

- Flowchart of the **do/while** repetition structure



Repetition Structure: do/while

```
1  /* Fig. 4.9: fig04_09.c
2     Using the do/while repetition structure */
3  #include <stdio.h>
4
5  int main()
6  {
7     int counter = 1;
8
9     do {
10        printf( "%d ", counter );
11    } while ( ++counter <= 10 );
12
13    return 0;
14 }
```

1. Initialize variable

2. Loop

3. Print

Program Output:

```
1 2 3 4 5 6 7 8 9 10
```

Multiple-Selection Structure: **switch**

- **switch**

- Useful when a variable or expression is tested for all the values it can assume and different actions are taken

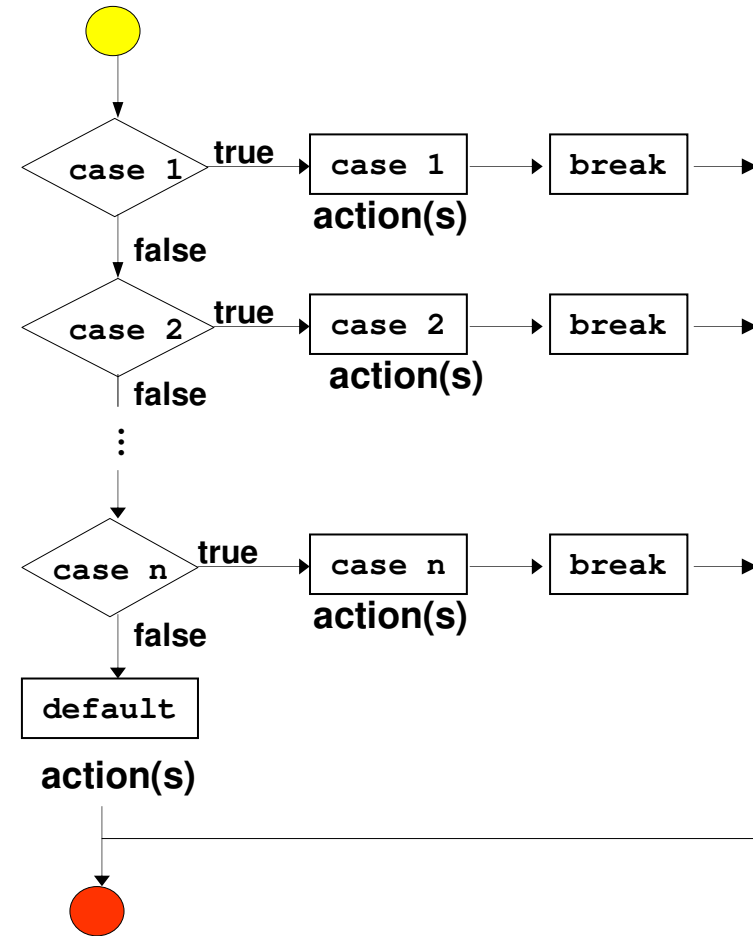
- **Format**

- Series of **case** labels and an optional **default** case

```
switch ( value ) {  
    case '1':  
        actions  
    case '2':  
        actions  
    default:  
        actions  
}
```

- **break**; exits from structure

- **Flowchart of the **switch** structure**



```

1  /* Fig. 4.7: fig04_07.c
2  Counting letter grades */
3  #include <stdio.h>
4
5  int main()
6  {
7  int grade;
8  int aCount = 0, bCount = 0, cCount = 0, dCount = 0, 9
9      fCount = 0;
10
11  printf( "Enter the letter grades.\n" );
12  printf( "Enter the EOF character to end input.\n" );
13
14  while ( ( grade = getchar() ) != EOF ) {
15
16      switch ( grade ) { /* switch nested in while */
17
18          case 'A': case 'a': /* grade was uppercase A */
19              ++aCount; /* or lowercase a */
20              break;
21
22          case 'B': case 'b': /* grade was uppercase B */
23              ++bCount; /* or lowercase b */
24              break;
25
26          case 'C': case 'c': /* grade was uppercase C */
27              ++cCount; /* or lowercase c */
28              break;
29
30          case 'D': case 'd': /* grade was uppercase D */
31              ++dCount; /* or lowercase d */
32              break;
33
34          case 'F': case 'f': /* grade was uppercase F */
35              ++fCount; /* or lowercase f */
36              break;
37

```

1. Initialize variables

2. Input data

3. Use switch loop to update count

```

38     case '\n': case ' ': /* ignore these in input */
39         break;
40
41     default: /* catch all other characters */
42         printf( "Incorrect letter grade entered." );
43         printf( " Enter a new grade.\n" );
44         break;
45     }
46 }
47
48 printf( "\nTotals for each letter grade are:\n" );
49 printf( "A: %d\n", aCount );
50 printf( "B: %d\n", bCount );
51 printf( "C: %d\n", cCount );
52 printf( "D: %d\n", dCount );
53 printf( "F: %d\n", fCount );
54
55 return 0;
56 }

```

4. Print results

Program Output:

```

Enter the letter grades.
Enter the EOF character to end input.
A
B
C
C
A
D
F
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
B

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1

```

The `break` and `continue` Statements

- `break`

- 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

- `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**
 - Increment expression is executed, then the loop-continuation test is evaluated

continue Statement

```
while (expr) {  
    statement  
    ...  
    continue;  
    statement  
    ...  
}  
  
do {  
    statement  
    ...  
    continue;  
    statement  
    ...  
} while (expr)
```

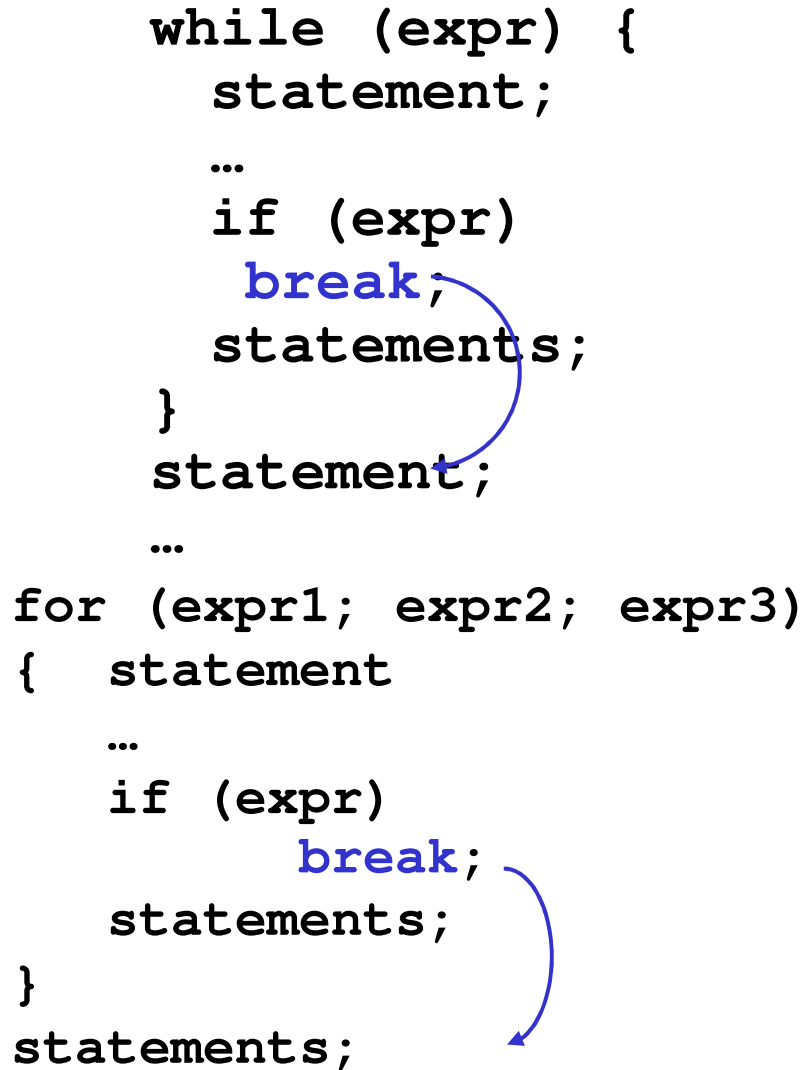
skip

```
for (expr1; expr2; expr3) {  
    statement  
    ...  
    continue;  
    statement  
    ...  
}
```

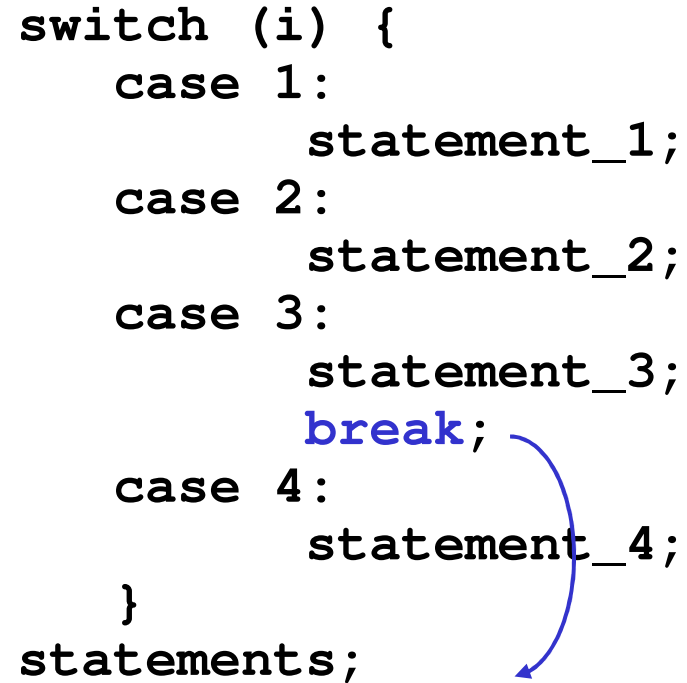
skip

break Statement

```
while (expr) {  
    statement;  
    ...  
    if (expr)  
        break;  
    statements;  
}  
statement;  
...  
for (expr1; expr2; expr3)  
{ statement  
...  
    if (expr)  
        break;  
    statements;  
}  
statements;
```



```
switch (i) {  
    case 1:  
        statement_1;  
    case 2:  
        statement_2;  
    case 3:  
        statement_3;  
        break;  
    case 4:  
        statement_4;  
}  
statements;
```



Equality (==) vs. 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: using ==:

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

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

Example: replacing == with =:

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

- This sets **paycode** to **4**
 - **4** is nonzero, so expression is **true**, and bonus awarded no matter what the **paycode** was
- Logic error, not a syntax error

Examples

Ex_1:

```
if (i=1) y = 3;
```

⇒ `y = 3` is always executed
this is not the same as

```
if (i==1) y = 3;
```

Ex_2:

```
if (i!=0) y=3;
```

⇒ `if (i) y=3;`

Ex_3:

```
if (i==0) y=3;
```

⇒ `if (!i) y=3;`

Examples:

Ex_1:

```
if (i>2)
    if (j==3)
        y=4;
    else
        y=5;
```

≠

```
if (i>2) {
    if (j==3)
        y=4;
}
else
    y=5;
```

=

```
if (i>2)
    if (j==3)
        y=4;
    else
        ;
else
    y=5;
```

Ex_2:

```
if (a>b)
    c = a;
else
    c = b;
```

⇒ `c = (a>b) ? a : b`

```
if (x==5)
    y = 1;
else
    y = 0;
```

⇒ `y = (x==5);`

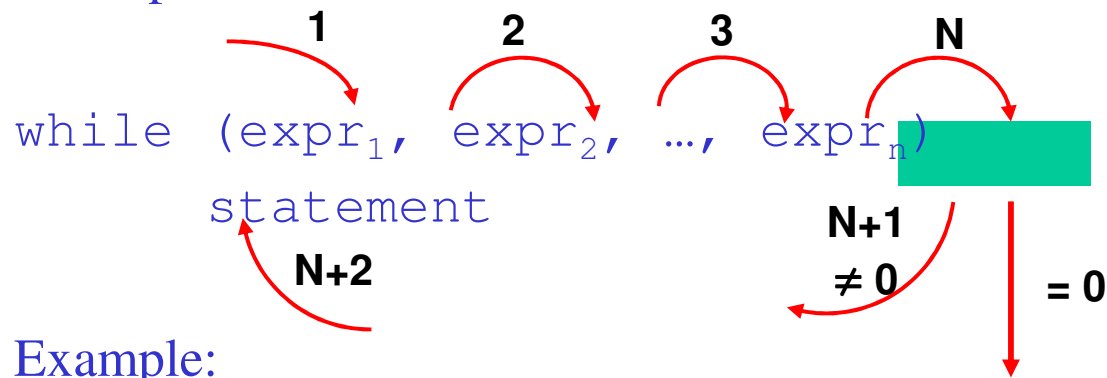
```
if (x<6)
    y = 1;
else
    y = 2;
```

⇒ `y = 2 - (x<6);`

⇒ `or y = 1 + (x>=6);`

Examples:

- **while** loop:



Example:

```
while (scanf("%d", &i), i--)  
    printf("%d", i);
```

- **Switch**

```
switch (i) {  
    case 1: j+=5;  
    case 2;  
    case 3: j+=4;  
    case 4: j+=3;  
}
```

The diagram shows four case labels: `i = 1`, `i = 2`, `i = 3`, and `i = 4`. Blue arrows point from each label to the corresponding case block in the switch statement: `case 1: j+=5;`, `case 2;`, `case 3: j+=4;`, and `case 4: j+=3;`.