Software and Embedded System Lab 2 (ELEE08022)

Making Decisions, Selection & Repetition in C language

Dr Jiabin Jia

Email: jiabin.jia@ed.ac.uk

Contents

- Repetition - *for* statement
- Repetition - *while* statement
- Make decision - *if-else* statement
- Make decision – *switch* statement

Combined Effect Operators

Assignment expressions which use the *same variable on both sides* of the assignment operator can be written using the combined operators:

`+=` `-=` `*=` `/=` `%=`

e.g. `sum += 12;` means `sum = sum + 12;`

`sum *= 12;` means `sum = sum * 12;`

The other part of the statement (`12` above) can be a much more complex expression

Combined Effect Operators

- We often want to add or subtract `1`

Incrementing	Decrementing
`++i;`	`--i;`
`i++;`	`i--;`

- preincrement/predecrement (`++i` and `--i`) variable’s value changed before value is used

```
 c = ++a + b;
        means
 a = a + 1; c = a + b;
```

- postincrement/postdecrement (`i++` and `i--`) variable’s value changed after value is used

```
c = a++ + b;
        means
c = a + b; a = a + 1;
```

The `for` statement

The *`for`* statement is a form of repetition.

It is especially useful in situations which use a fixed count condition.

The general form of the `for` statement is:

```
for (initializing list; tested expression; altering list)
      statement;
```

An example of a simple `for` statement is:

```
for (i = 5; i <= 15; i += 2)
    printf("%d ", i);
```

```
Output: 5 7 9 11 13 15
```

```
/*Example, print a table of square root values */
#include <stdio.h>
#include <math.h>
int main(void) /* void means no arguments for main() */
{
      int count;
      printf("NUMBER SQUARE ROOT\n");
      printf("------ -----------\n");

      for (count = 1; count <= 5; count += 1)
        {
            printf(" %d    %f\n", count, sqrt(count));
        }
      return 0;
}
```

```
NUMBER SQUARE ROOT
------ -----------
 1      1.000000
 2      1.414214
 3      1.732051
 4      2.000000
 5      2.236068
```

Mathematical Library Functions

C provides many standard pre-programmed functions which may be used in any program

- just like `printf( )` **and** `scanf( )`

To use the mathematical ones, we must know:

- Include math function header

```
#include <math.h>
```

- The **name** of the mathematical function
- The **type** of **input** data required by the C function
- The **type** of the **result** returned by the C function

Some Math Functions

Function Name	*argument*	*result*	*Description*
`abs(i)`	integer	integer	Absolute value of `i`
`fabs(d)`	double	double	Absolute value of `d`
`pow(d1, d2)`	double	double	`d1` raised to the `d2` power
`exp(d)`	double	double	`e` raised to the `d` power
`sqrt(d)`	double	double	Square root of `d`
`sin(d)`	double	double	Sine of `d` (`d`in radians)
`cos(d)`	double	double	Cosine of `d` (`d` in radians)
`tan(d)`	double	double	Tangent of `d` (`d` in radians)
`tanh(d)`	double	double	Hyperbolic tangent of `d`
`log(d)`	double	double	Natural log of `d`
`log10(d)`	double	double	Common log (base 10) of `d`

For example :-

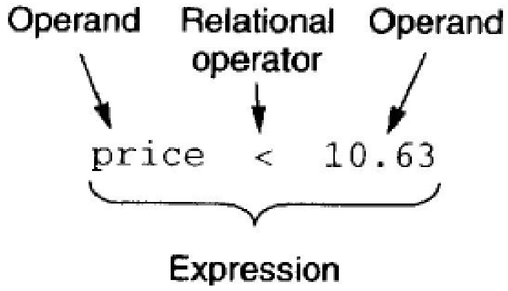
```
cubetwo = pow(2.0, 3.0);

printf("%f\n", sqrt(4.0));
```

Relational Expressions

some operators perform comparisons of values and give logical (true/false) result

- Relational operators:



Operator	**Meaning**	**Example**
`<`	less than	`age < 21`
`>`	greater than	`height > 5.6`
`<=`	less than or equal to	`speed <= 30.0`
`>=`	greater than or equal to	`mark >= 40`
`==`	equal to	`price == 5.99`
`!=`	not equal to	`cash != 0.0`

the following are ***invalid*** :

`length =< 50` *(operator symbols out of order)*

`flag = = done` *(spaces not allowed within operator)*

Logical Operators

`&&`	(AND)
`||`	(OR)
`!`	(NOT)

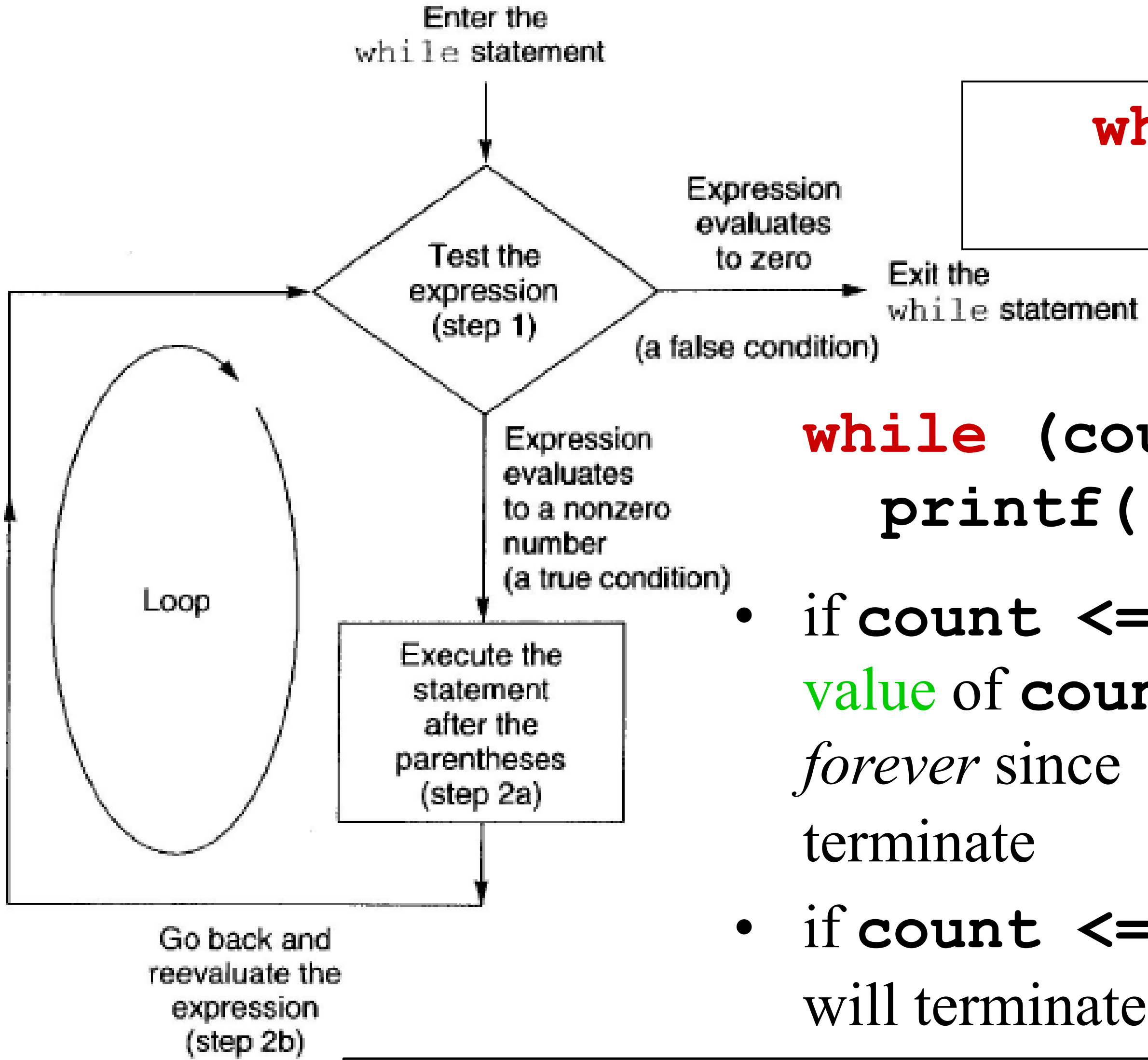
- *logical expressions* act as we should expect from English

  ```
  (voltage > 40.0) && (current < 10.0)
  ```

  is *true* (produces the value `1`) only if both expressions inside parentheses are true.

- the *unary* NOT operator `!` is used to change any logic expression to its opposite condition (logical inversion)

The while statement


Enter the while statement
Test the expression (step 1)
Expression evaluates to zero
Exit the while statement
(a false condition)
Expression evaluates to a nonzero number
(a true condition)
Loop
Execute the statement after the parentheses (step 2a)
Go back and reevaluate the expression (step 2b)

```
while (expression)
    statement;
```

```
while (count <= 10)
  printf("%d ", count);
```

- if `count <= 10` is true, the same value of `count` will be printed out *forever* since **while** will never terminate
- if `count <= 10` is false, **while** will terminate *without* printing anything

```
while (1)
  statement;
```

statement is executed indefinitely.

while statement is alternative to *for* statement:

```
#include <stdio.h>
#include <math.h>
int main(void)
{
   int count;

   printf("NUMBER SQUARE ROOT\n");
   printf("------ -----------\n");

   count = 1;
   while (count <= 5)
   {
     printf(" %d     %f\n", count, sqrt(count));
     count += 1;
   }
   return 0;
}
```

```
/*Input 4 numbers and calculate their sum and average*/
#include <stdio.h>
int main(void)
{
      int count;
      float num, total;

      count = 0;      /* nothing read to start with */
      total = 0.0;
      while (count < 4)
       { /* count runs from 0 to 3 */
            scanf("%f", &num);
            total += num;
            ++count;
       }
      printf("\nTotal is %6.2f, Average is %4.2f\n",
                               total, total / count);
      return 0;
}
```

```
  input: 45 67 34 87
  output: Total is 233.00, Average is 58.25
```

The break Statement

A break statement forces an immediate exit (break out) from repetition loops, regardless of the value of the loop control variable (it is also an important partner of most switch statements)

e.g. immediately terminate loop if a value < 0 or > 100 is read:

```
while (count < 4)
{
   scanf("%f", &num);
   if (num < 0.0 || num > 100.0)
   {
      printf("Error: Invalid Mark %f\n", num);
      break;                /* break out of the loop */
   }
   total += num;
   ++count;
}
/* execution continues here immediately after break */
```

Useful for terminating loops when an *unusual condition* is detected

The continue Statement

continue is similar to break, but applies only to repetition loops, but not switches. When continue is encountered, the next iteration of the loop begins immediately.

e.g. continue here ignores invalid marks:

```
while (count < 4)
{
  scanf("%f", &num);
  if (num < 0.0 || num > 100.0)
  {
    printf("Error: Invalid Mark %f\n", num);
    continue;   /* skip rest of loop body */
  }
  total += num;
  ++count;
  /* continue arrives here - go straight back to top */
}
```

Useful for omitting invalid input, while staying in the loop!

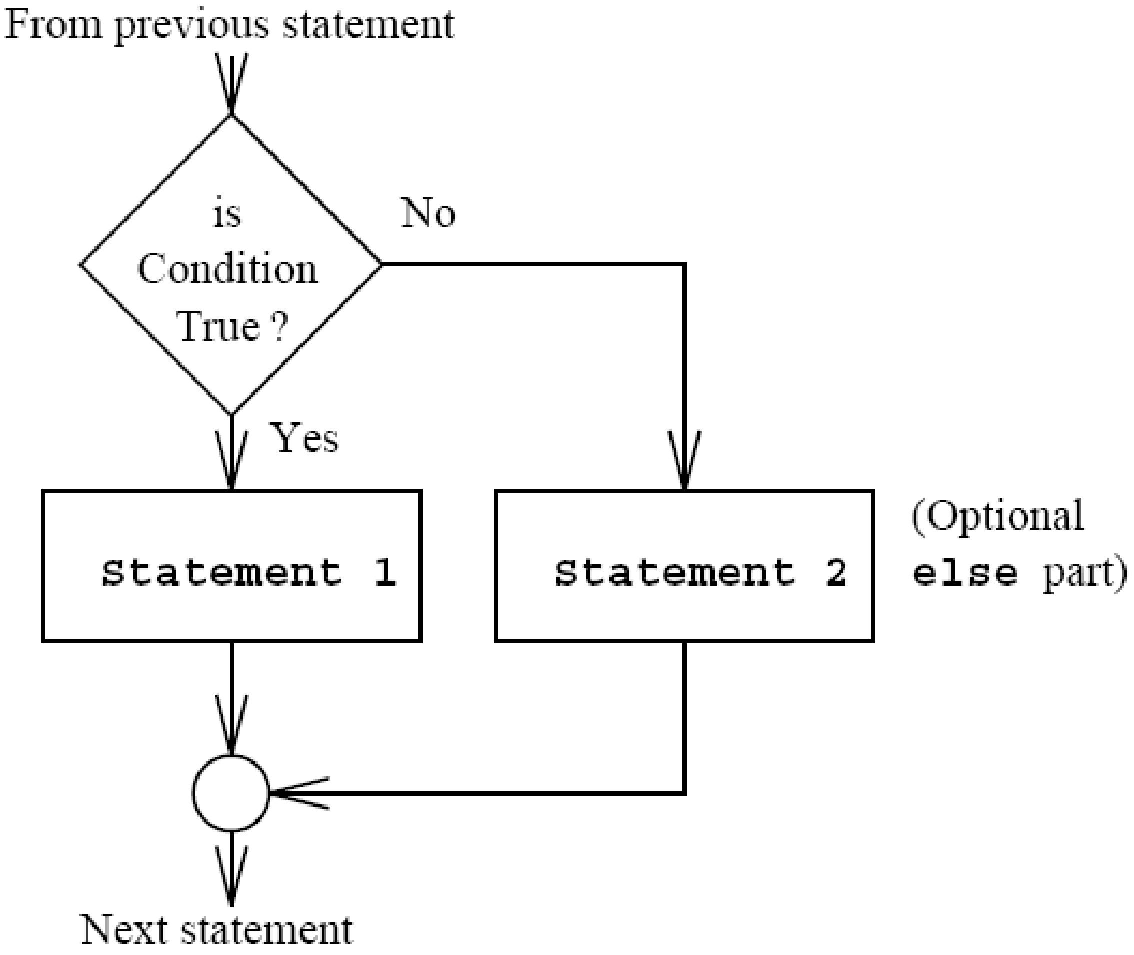
Making Decisions

- *control-flow* statements allow us to select or change the order in which computations are performed

- the ***if-else*** construct is used to *make decisions*

```
if (condition)
    statement 1;
else
    statement 2;
```

- *semicolons terminate **statement**(s) only!*



A Flowchart for the if-else statement

```c
/* Read a value of current from keyboard
 * and decide if it would blow a 3A fuse. */

#include <stdio.h>
int main(void)
{
  float current;

  scanf("%f", &current);

  /* compare current against fuse rating */
  if (current >= 3.0)
    printf("Warning: Fuse will blow\n");
  else
    printf("Fuse should not blow\n");

  return 0;
}
```

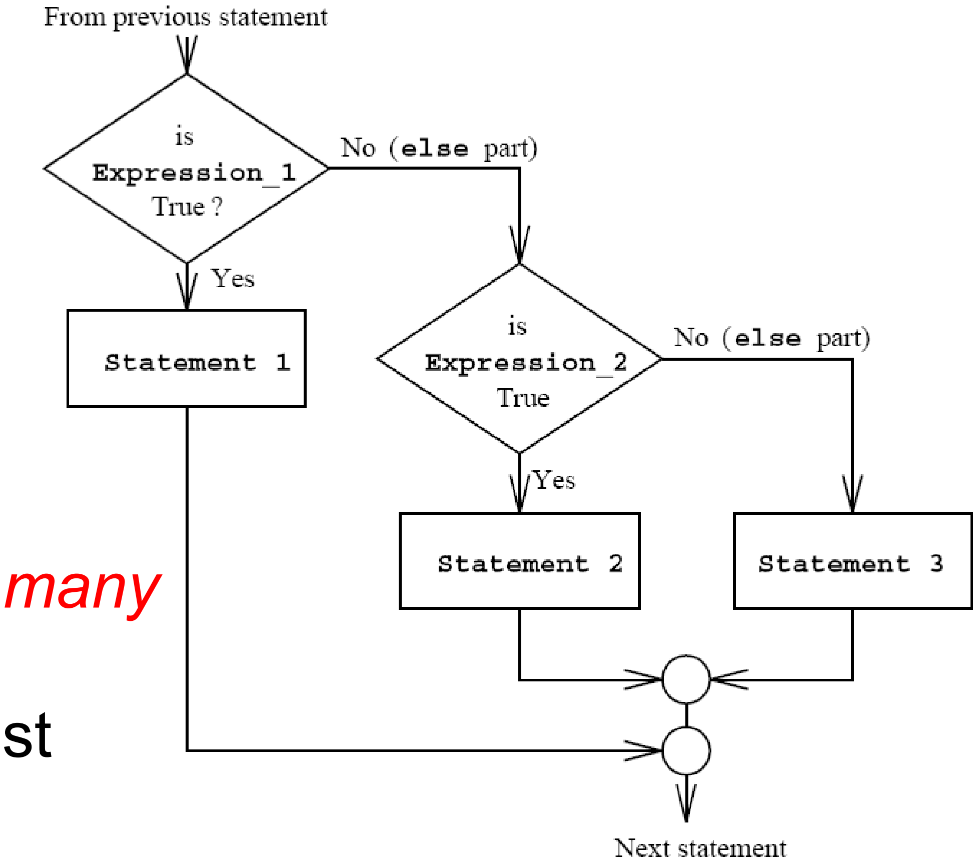
```
Input: 5.0
Warning: Fuse will blow

Input: 2.5
Fuse should not blow
```

Complex Decision Making

- A sequence of **if – else if - … - else** statements is the most general way of writing a multi-way decision

```
if (expression_1)
   statement_1;
else if (expression_2)
   statement_2;
else
   statement_3;
```



- single *statement_n* may be *many statements* in braces { }
- **else** makes pair with closest preceding unpaired **if**

```c
/* Given power drawn by mains appliance, select a suitable fuse from
 * standard values, assuming rated current can be carried indefinitely.
 */
#include <stdio.h>
int main(void)
{
  float power, current;

  scanf("%f", &power);     /* read in power consumption */
  current = power / 230.0;  /* use nominal line voltage */
  printf("Current drawn at 230V is %6.2fA\n", current);
  /* order of tests significant: used to create ranges */
  if (current >= 13.0)
    printf("Device unsuitable for 13A plug!\n");
  else if (current >= 5.0)
    printf("Use a 13A fuse for this appliance\n");
  else if (current >= 3.0)
    printf("Use a 5A fuse for this appliance\n");
  else if (current >= 0.0)
    printf("Use a 3A fuse for this appliance\n");
  else
    printf("ERROR: NEGATIVE current ??!!\n");
  return 0;
}
```

The switch Statement

- **if-else** structures select one set of instructions from many possible alternatives, based on *simple or complex conditions*
- **switch** statement is alternative for situation where **single integer expression** can generate values which distinguish alternatives
- The general form is:

```
switch (expression) /* expression is NOT logical */
{                   /* start of compound statement */

  /* case order not significant */
  case value_1:     /* case selector ends with a colon */
     statement1;
     statement2;
     break;         /* end of statements for this case */

  case value_n:
     statementw;
     statementx;
     break;

  default:         /* all-other-values case */
     statementaa;
     statementbb;
     break;
}           /* end of compound statement and switch */
```

```c
#include <stdio.h>     /* Switch statement version */
#define FAHR 'F'  /* Indicates a Fahrenheit temperature */
#define CENT 'C'  /* Indicates a Centigrade temperature */

int main(void)
{
  float in_temp, out_temp;
  char  type;

  scanf("%f %c", &in_temp, &type);  /* read in temperature */
  switch (type)           /* select by character value */
   {
    case CENT:            /* Centigrade conversion required */
      out_temp = in_temp * (9.0 / 5.0) + 32.0;
      printf("%6.2f deg C = %6.2f deg F\n", in_temp, out_temp);
      break;

    case FAHR:            /* Fahrenheit conversion required */
      out_temp = (5.0 / 9.0)*(in_temp - 32.0);
      printf("%6.2f deg F = %6.2f deg C\n", in_temp, out_temp);
      break;

    default:       /* catch anything other than 'C' or 'F' */
      printf("Error in data - don’t understand %c\n", type);
      break;
  }
  return 0;
}
```