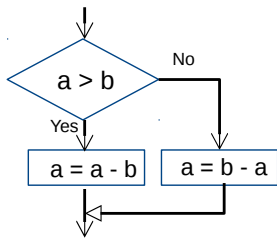


# C Programming

## Sequential and Conditional Control



Lecturer: *Dr. Wan-Lei Zhao*  
*Autumn Semester 2022*

- 1 Editors and IDE
- 2 Basic Ingredients of a C Program
- 3 Program structure
- 4 Program of Sequential Control
- 5 Conditional Controls
  - if-else clause
  - Logical operators
- 6 Conditions

- Integrated Development Environment
  - VScode (default option)
  - Dev-C++ (recommended)
  - Codeblocks
- Any text editors + C compiler
  - Microsoft C compiler (default option)
  - GCC (recommended)
  - ICC: believed to be the most efficient one
  - Turbo C: classic but being forgotten

- 1 Editors and IDE
- 2 Basic Ingredients of a C Program
- 3 Program structure
- 4 Program of Sequential Control
- 5 Conditional Controls
  - if-else clause
  - Logical operators
- 6 Conditions

# The first program

- Create a new file named **main.c**.
- Open it in your text editor of choice.
- Fill it as follows:

```
1 #include <stdio.h>
2 int main(void)
3 {
4     printf("Hello-World!\n");
5     /* Print "Hello World!" on the command line */
6     return 0;
7 }
8
```

# From source to bits

Source code: main.c



Linux/Mac OS X

```
gcc main.c -o hello
```

Windows

```
D:\mycode\gcc main.c -o hello
```

(Preprocessing → compiling → assembling → linking)



Executable program

Linux/Mac OS X (**hello**)

```
$ ./hello  
$ Hello World!
```

Windows (**hello.exe**)

```
D:\mycode\hello  
Hello World!  
D:\mycode\
```

- 1 Editors and IDE
- 2 Basic Ingredients of a C Program
- 3 Program structure**
- 4 Program of Sequential Control
- 5 Conditional Controls
  - if-else clause
  - Logical operators
- 6 Conditions

# A basic program

```
1 #include <stdio.h>
2 int main()
3 {
4     printf(" Hello-World!\n" );
5     /* Print "Hello World!" on the
6        command line */
7     return 0;
8 }
```

} Preprocessing statements

} Main function

- Processed before compilation
- Have their own language, start with a #
- In 'stdio.h', function '**printf()**' has been defined



# The main function

- Basic function of every program
- Exists **exactly once** per program
- Called on program start

```
1  int main(void)
2  {
```

- As a function, *main()* can take parameters and return a value
- Get used to *void* and *int*. They will be explained later
- '{' marks the start of the main function scope

# The main function scope

- Contains program statements
- They are processed from top to bottom

```
1  return 0;  
2 }
```

- Last statement, ends main function (and thus the whole program)
- `0` tells the OS that everything went right
- `'}'` marks the end of the main function scope

# Statements

- Instructions for the computer
- End with a ; (semicolon)

```
1 printf(" Hello -World!\n" );
```

- Here is the empty statement:

```
;
```

- All statements are located in function blocks

# Comments

- Information for the programmer, cut out before compilation

Single line comments:

```
1 // Prints "Hello World!" on the command line
```

Block comments (multi-line):

```
1 /* Prints "Hello World!"  
2    on the command line */
```

Better style of block comments:

```
1 /*  
2  * Prints "Hello World!"  
3  * on the command line  
4  */
```

# Order of execution

- Statements inside one function executed from top to bottom
- This is a convention for languages

```
1 #include <stdio.h>
2 int main()
3 {
4     printf(" Hello - China!\n" );
5     printf(" Hello - World!\n" );
6     printf(" Hello - Universe!\n" );
7     return 0;
8 }
```

```
1 Hello China!
2 Hello World!
3 Hello Universe!
4
```

- For **clarity**, one statement in one line

- 1 Editors and IDE
- 2 Basic Ingredients of a C Program
- 3 Program structure
- 4 Program of Sequential Control**
- 5 Conditional Controls
  - if-else clause
  - Logical operators
- 6 Conditions

# Calculate the Area of a circle (1)

- Available information
  - radius,  $\pi = 3.1415$
- Requirements
  - Allows user input radius of a circle
  - Calculate its area and print it out

$$a = \pi \cdot r^2$$

- Let's do it step by step

## Calculate the Area of a circle (2)

- Create a new file named **main.c**
- Open it in your editor
- Fill it as follows:

```
1 #include <stdio.h>
2 int main()
3 {
4     return 0;
5 }
6
```



# Calculate the Area of a circle (3)

- Define variables needed

```
1 #include <stdio.h>
2 int main()
3 {
4     float pi = 3.1415;
5     float r = 0;
6     return 0;
7 }
8
```

# Calculate the Area of a circle (4)

- Allows user to input radius,

```
1 #include <stdio.h>
2 int main()
3 {
4     float pi = 3.1415;
5     float r = 0;
6     scanf("%f", &r);
7     return 0;
8 }
9
```

# Calculate the Area of a circle (5)

- Allows user to input radius

```
1 #include <stdio.h>
2 int main()
3 {
4     float pi = 3.1415;
5     float r = 0, area = 0;
6     scanf("%f", &r);
7     area = r*r*pi;
8     return 0;
9 }
10
```

- The complete program

```
1 #include <stdio.h>
2 int main()
3 {
4     float pi = 3.1415;
5     float r = 0, area = 0;
6     scanf("%f", &r);
7     area = r*r*pi;
8     printf("Area: %f", area);
9     return 0;
10 }
11
```

# Solve Quadratic Equation (1)

- Given following equation
- Allows user input  $a$ ,  $b$  and  $c$

$$ax^2 + bx + c = 0$$

- Solve  $x$  out

## Solve Quadratic Equation (2)

- The solution for this quadratic equation is well-known
- Given  $b^2 - 4ac > 0$ , we have

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$
$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

- In order to simplify the calculation
- We have

$$p = \frac{-b}{2a}, \quad q = \frac{\sqrt{b^2 - 4ac}}{2a}$$
$$x_1 = p + q, \quad x_2 = p - q$$

# Solve Quadratic Equation (3)

- Let's now think about how to implement it in C

$$p = \frac{-b}{2a}, \quad q = \frac{\sqrt{b^2 - 4ac}}{2a}$$
$$x_1 = p + q, \quad x_2 = p - q$$

- Define variables and user input

```
1 #include <stdio.h>
2 int main()
3 {
4     float a = 0, b = 0, c = 0, delta = 0;
5     float x1 = 0, x2 = 0, p = 0, q = 0;
6     printf("Input -a, -b and -c:\n");
7     scanf("%f%f%f", &a, &b, &c);
8     return 0;
9 }
10
```

# Solve Quadratic Equation (4)

$$p = \frac{-b}{2a}, \quad q = \frac{\sqrt{b^2 - 4ac}}{2a}$$

$$x_1 = p + q, \quad x_2 = p - q$$

```
1 #include <stdio.h>
2 #include <math.h>
3 int main()
4 {
5     float a = 0, b = 0, c = 0, delta = 0;
6     float x1 = 0, x2 = 0, p = 0, q = 0;
7     printf("Input a, b and c:\n");
8     scanf("%f%f%f", &a, &b, &c);
9     delta = b*b - 4*a*c;
10    p = -b/(2*a);
11    q = sqrt(delta)/(2*a);
12    x1 = p + q; x2 = p - q;
13    printf("x1=%f, x2=%f\n", x1, x2);
14    return 0;
15 }
16
```

$$p = \frac{-b}{2a}, \quad q = \frac{\sqrt{b^2 - 4ac}}{2a}$$
$$x_1 = p + q, \quad x_2 = p - q$$

- In above example, we did not consider the case
- $b^2 - 4ac < 0$
- For which, we should output “no real solution”
- That means, we should check  $b^2 - 4ac$
- For different case, we give different answer
- This is where `if...else` fits in



# Outline

- 1 Editors and IDE
- 2 Basic Ingredients of a C Program
- 3 Program structure
- 4 Program of Sequential Control
- 5 Conditional Controls**
  - if-else clause
  - Logical operators
- 6 Conditions

- 1 Editors and IDE
- 2 Basic Ingredients of a C Program
- 3 Program structure
- 4 Program of Sequential Control
- 5 Conditional Controls**
  - if-else clause
  - Logical operators
- 6 Conditions

# Start with a simple example

- Guess what the following code for

```
1 #include <stdio.h>
2 int main()
3 {
4     int x = 5;
5     if(x%2 == 0)
6     {
7         printf("x is an even number.");
8     }
9     return 0;
10 }
11
```

- `if` statement makes a judgement
- If the **logic/conditional** expression is **true**
- The statment(s) inside `{...}` will be executed
- Otherwise, statment(s) will be **ignored**

# Logic/conditional expression (1)

- Let's now focus on the conditional expression

```
1 #include <stdio.h>
2 int main()
3 {
4     int x = 5;
5     if(conditional_expression)
6     {
7         printf("x is an even number.");
8     }
9     return 0;
10 }
11
```

- It is an expression that returns **true** or **false**
- For example, statement “you are undergraduate student”
- We can judge whether it is true or false
- Paradox**, story shared

## Logic/conditional expression (2)

- In C, expression with **relational operators** is used as conditional expressions
- They are
  - ① `<`, `>`, `<=`, `>=`
  - ② `==` for “equal to”
  - ③ `!=` for “not equal to”
- It returns 1 (true) or 0 (false)

```
1 int main()  
2 {  
3     int a = 0, b = 0, c = 0;  
4     a = (3 > 5);  
5     b = (2*2 > 4);  
6     c = (3 == 3);  
7     return 0;  
8 }
```

$$p = \frac{-b}{2a}, \quad q = \frac{\sqrt{b^2 - 4ac}}{2a}$$
$$x_1 = p + q, \quad x_2 = p - q$$

- For the case  $b^2 - 4ac < 0$
- We should output “no real solution”
- For the case,  $b^2 - 4ac \geq 0$
- We should output  $x_1$  and  $x_2$

# Solve Quadratic Equation

```
1 #include <stdio.h>
2 #include <math.h>
3 int main()
4 {
5     float a = 0, b = 0, c = 0, delta = 0;
6     float x1 = 0, x2 = 0, p = 0, q = 0;
7     printf("Input a, b and c:\n");
8     scanf("%f%f%f", &a, &b, &c);
9     delta = b*b - 4*a*c;
10    if(delta >= 0){
11        p = -b/(2*a);
12        q = sqrt(delta)/(2*a);
13        x1 = p + q; x2 = p - q;
14        printf("x1=%f, x2=%f\n", x1, x2);
15    } else {
16        printf("No real solution!\n");
17    }
18    return 0;
19 }
```

# Exercise

```
1 #include <stdio.h>
2 int main()
3 {
4     int a = -1;
5     unsigned int b = 600;
6     if(a > b)
7     {
8         printf("%d is greater than %d\n", a, b);
9     } else {
10        printf("%d is smaller than %d\n", a, b);
11    }
12    return 0;
13 }
14
```

- What is the answer??



- 1 Editors and IDE
- 2 Basic Ingredients of a C Program
- 3 Program structure
- 4 Program of Sequential Control
- 5 Conditional Controls**
  - if-else clause
  - Logical operators
- 6 Conditions

# Logical Operators (1)

- In some cases, single conditional statement is not enough
- For example, we want to express following condition
- If  $a > b$  AND  $b > c$ , then ...
- We need a way to connect several statements
- Usually, we use AND, OR and NOT
- In C, they are `&&`, `||` and `!`
- AND ( $c1 \ \&\& \ c2$ ): means only when  $c1$  and  $c2$  both are true, it is true
- OR ( $c1 \ || \ c2$ ): means when either  $c1$  or  $c2$  is true, it is true
- NOT ( $!c1$ ): means reverse it,  $c1$  is true,  $!c1$  is false;  $c1$  is false,  $!c1$  is true

## Logical Operators (2)

- AND ( $c1 \ \&\& \ c2$ ): means only when  $c1$  and  $c2$  both are true, it is true
- OR ( $c1 \ || \ c2$ ): means when either  $c1$  or  $c2$  both is true, it is true
- NOT ( $!c1$ ): means reverse it,  $c1$  is true,  $!c1$  is false;  $c1$  is false,  $!c1$  is true

```
1 int main()  
2 {  
3     int a = 0, b = 0, c = 0;  
4     a = (3 > 5) && (2 > 1);  
5     b = (2*2 > 4) || (2 == 1);  
6     c = !(3 == 3);  
7     return 0;  
8 }
```

## Logical Operators (3): truth tables

c1	c2	c1 && c2
1	1	1
1	0	0
0	1	0
0	0	0

c1	c2	c1    c2
1	1	1
1	0	1
0	1	1
0	0	0

c1	!c1
1	0
0	1

- One should be able to deduce for cases that more than two statements are involved

# Priority of Relational Operators (1)

1 5>3 && 2 || 8<4-!0

- “!” is higher than +, -, \*, /
- “>”, “<”, “<=”, “>=” and “==” are lower than +, -, \*, /
- “&&” and “||” are the lowest
- Please tell me the result of this expression (**3 minutes**)

## Priority of Relational Operators (2)

```
1 5>3 && 2 || 8<4-!0
2 5>3 && 2 || 8<4-1
3 5>3 && 2 || 8<3
4 5>3 && 2 || 0
5 1 && 2 || 0
6 1 || 0
7 1
```

- “!” is higher than +, -, \*, /
- “>”, “<”, “<=”, “>=” and “==” are lower than +, -, \*, /
- “&&” and “||” are the lowest

## Priority of Relational Operators (3)

1 (5>3) && 2 || (8<(4-!0))

- Usually, we put '()' to regularize the priority levels

# How the logic expression is evaluated in C

- C actually checks only whether it is zero or non-zero
- For example

```
1 int main()  
2 {  
3     float a = 3.1, b = 0;  
4     if(a){  
5         printf("it-is-true");  
6     } else {  
7         printf("it-is-false");  
8     }  
9     if(a && b){  
10        printf("it-is-true");  
11    } else {  
12        printf("it-is-false");  
13    }  
14    return 0;  
15 }
```



- 1 Editors and IDE
- 2 Basic Ingredients of a C Program
- 3 Program structure
- 4 Program of Sequential Control
- 5 Conditional Controls
  - if-else clause
  - Logical operators
- 6 Conditions

# if...else

Decisions are made during run time:

```
if(condition)
    statement1;
else
    statement2;
```

**statement1** is only executed if the truth value of **condition** is *true*.  
Otherwise **statement2** is executed.

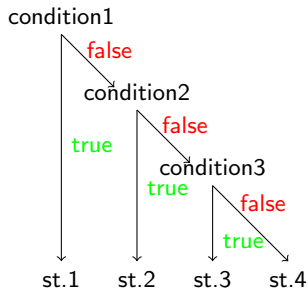
For multiple statements inside the **if-else**, use braces {}:

```
if(condition) {
    statement1;
    statement2;
}
```

- The **else** part is OPTIONAL

# else if

To differentiate between more than two cases, you can use the if condition as a statement in the else body:



```
if (condition1)
    statement1;
else if (condition2)
    statement2;
else if (condition3)
    statement3;
else
    statement4;
```

# Judge the Type of an Input Character (1)

- Judge an input character is a digit, a character, space or something else
- Steps outlined
  - ① Accept/take input character
  - ② Check whether in digit range ('0'-'9')
  - ③ Otherwise, check whether it is in character range ('a'-'z')
  - ④ Otherwise, check whether it is in ('A'-'Z')
  - ⑤ Otherwise, check whether it is space (' ')
- Work it by yourself first...

## Judge the Type of an Input Character (2)

```
1 #include <stdio.h>
2 int main()
3 {
4     char ch = ' ';
5     printf("Please input a character: ");
6     ch = getchar();
7     printf("Character is: %c", ch);
8     if(ch >= '0' && ch <= '9'){
9         printf("a digit\n");
10    }else if(ch >= 'a' && ch <= 'z'){
11        printf("char in lower case\n");
12    }else if(ch >= 'A' && ch <= 'Z'){
13        printf("char in upper case\n");
14    }else if(ch == ' '){
15        printf("It is space\n");
16    }else{
17        printf("not digit, char or space\n");
18    }
19    return 0;
20 }
```

# Judge whether it is a leap year (1)

- Leap year should satisfy one of follow two conditions
  - ① It is dividable by 4, but not by 100
  - ② It is dividable by 400

[Steps]

- ① Accept input number
- ② Check whether it is dividable by 400
- ③ If yes, it is leap year
- ④ Otherwise, check whether it is dividable by 4 and NOT dividable by 100
  - ① If yes, it is leap year
  - ② Otherwise, it is not leap year

Give your solution first....

## Judge whether it is a leap year (2)

```
1 #include <stdio.h>
2 int main()
3 {
4     int year = 0, leap = 0;
5     printf(" Please-enter-the-year:-");
6     scanf("%d", &year);
7     if(year%400 == 0){
8         leap = 1;
9     }else if(year%4 == 0 && year%100 != 0){
10         leap = 1;
11     }else{
12         leap = 0;
13     }
14     if(leap == 1){
15         printf("%d-is-leap-year\n", year);
16     }else{
17         printf("%d-is-not-leap-year\n", year);
18     }
19 }
```

# More about if-else clause (1)

- See the result of following code

```
1 int main()  
2 {  
3     int a = 3, b = 5, c = 3;  
4     if(a != 3)  
5     if(b > 9)  
6         printf("b==%d", b);  
7     else  
8         printf("c==%d", c);  
9     return 0;  
10 }
```



## More about if-else clause (2)

- See the result of following code

```
1 int main()  
2 {  
3     int a = 3, b = 5;  
4     int c = 3;  
5     if(a != 3)  
6     if(b > 9)  
7         printf("b==%d", b);  
8     else  
9         printf("c==%d", c);  
10    return 0;  
11 }
```

```
1 int main()  
2 {  
3     int a = 3, b = 5, c = 3;  
4     if(a != 3)  
5     {  
6         if(b > 9)  
7             printf("b==%d", b);  
8         else  
9             printf("c==%d", c);  
10    }  
11    return 0;  
12 }
```

# A few words on style

- Do not put statements and conditions on the same line

```
if(cond){ statement; } /* bad style */
```

```
if(cond){          /* looks better , still bad style */  
    statement;  
}
```

```
if(cond)  
    statement; /* It is OK but not recommended , put {} all  
               the time */
```

# More words on style

- Inside an `if-else` structure
- Put all blocks of this structure in braces

```
if(cond)           /* bad style , inconsistent */
    statement;
else {
    statement;
    statement;
}
```

```
if(cond){
    /* way better style */
    statement;
} else {
    statement;
    statement;
}
```

```
if (cond)
{
    statement;
} else
{
    statement;
    statement;
}
```

# Operator: $L=a*b?c:d$

- Following codes produce the same results

```
1 int main()  
2 {  
3     int a = 3, b = 4, c = 1;  
4     if(a > b)  
5     {  
6         a = c;  
7     } else {  
8         a = b;  
9     }  
10    printf("a=%d", a);  
11 }
```

```
1 int main()  
2 {  
3     int a = 3, b = 4, c = 1;  
4     a = a > b ? c : b;  
5     printf("a=%d", a);  
6 }
```

- " $a*b$ " is a logic expression
- If it is true, c is assigned to the left
- Otherwise, d is assigned to the left

# Application: Convert lower case char to upper case (1)

- Given a char of unknown case, convert it to uppercase
- 'a'-'z' to 'A'-'Z'
- Solution:
  - 1 Check whether **ch** is in the range of 'a'-'z'
  - 2 If it is in this range,  $ch = ch - 32$
  - 3 Otherwise, do not do anything

## Application: Convert lower case char to upper case (2)

```
1 int main()  
2 {  
3     char ch = getchar();  
4     if(ch >= 'a' && ch <= 'z')  
5     {  
6         ch = ch - 32;  
7     }  
8     printf("ch=%c", ch);  
9     return 0;  
10 }
```

```
1 int main()  
2 {  
3     char ch = getchar();  
4     ch = (ch >= 'a' && ch <= 'z')  
5         ? (ch - 32) : ch;  
6     printf("ch=%c", ch);  
7     return 0;  
8 }
```

- It is concise
- Do not make your expression too long
- Take the left way when you are uncertain

## if clause: the last example

```
1 int main()  
2 {  
3     int a = 3, b = 5, c = 2;  
4     if( a > b);  
5     {  
6         a = c;  
7     }  
8     a = a*2;  
9     printf("a = %d\n", a);  
10    return 0;  
11 }
```

# switch-case clause (1)

- Now you are given a new task
- Convert numbers (1-12) to Month (January - December)
- We can do it by **if-else** clause

```
int main()
{
    int n = 0;
    scanf("%d", &n);
    if(n == 1)
    {
        printf("January\n");
    } else if(n == 2){
        printf("Febuary\n");
    } else if(n == 3){
        printf("March\n");
    }
    ...
    ...
    ...
}
```



## switch-case clause (2)

- Now you are given a new task
- Convert numbers (1-12) to Month (January - December)
- We can do it by **if-else** clause

```
int main()
{
    int n = 0;
    scanf("%d", &n);
    switch(n)
    {
        case 1: {printf("January\n"); break;}
        case 2: {printf("Febuary"); break;}
        case 3: {printf("March\n"); break;}
        case 12: {printf("December\n"); break;}
    }
    return 0;
}
```

## switch-case clause (3)

- If you have to check one variable for many constant values
- `switch-case` is your friend:)

```
switch (variable)
{
    case option1: statement1; break;
    case option2: statement2; break;
    case option3: statement3; break;
    default: statement4; break;
}
```

- *case option* defines a jump label
- More than one statement after it possible without braces
- All statements until the next `break;` will be executed

## switch-case clause (4)

- What **break** means
- Work out the output of following codes

```
int main()
{
    int n = 3;
    switch(n)
    {
        case 1: { printf("January\n"); }
        case 2: { printf("Febuary"); break;}
        case 3: { printf("March\n"); }
        case 4: { printf("April\n"); }
        case 5: { printf("May\n"); break;}
        case 12: { printf("December\n"); break;}
        default: break;
    }
    return 0;
}
```