Fundamentals of Programming and Algorithm

C skeleton

```c
#include <stdio.h>

int main()
{

}
```

- All programs need this skeleton

Hello World

```c
#include <stdio.h>

int main()
{
   printf("Hello World!\n");
   printf("I'm 18 today!\n");
}
```

- Code of an imperative progamming languages is a series of commands that ask the computer to do something for you
- Those commands are in lines
- They are going to be executed in written order
- `;` semicolon ends a line of the code, but new line doesn't

Bugs

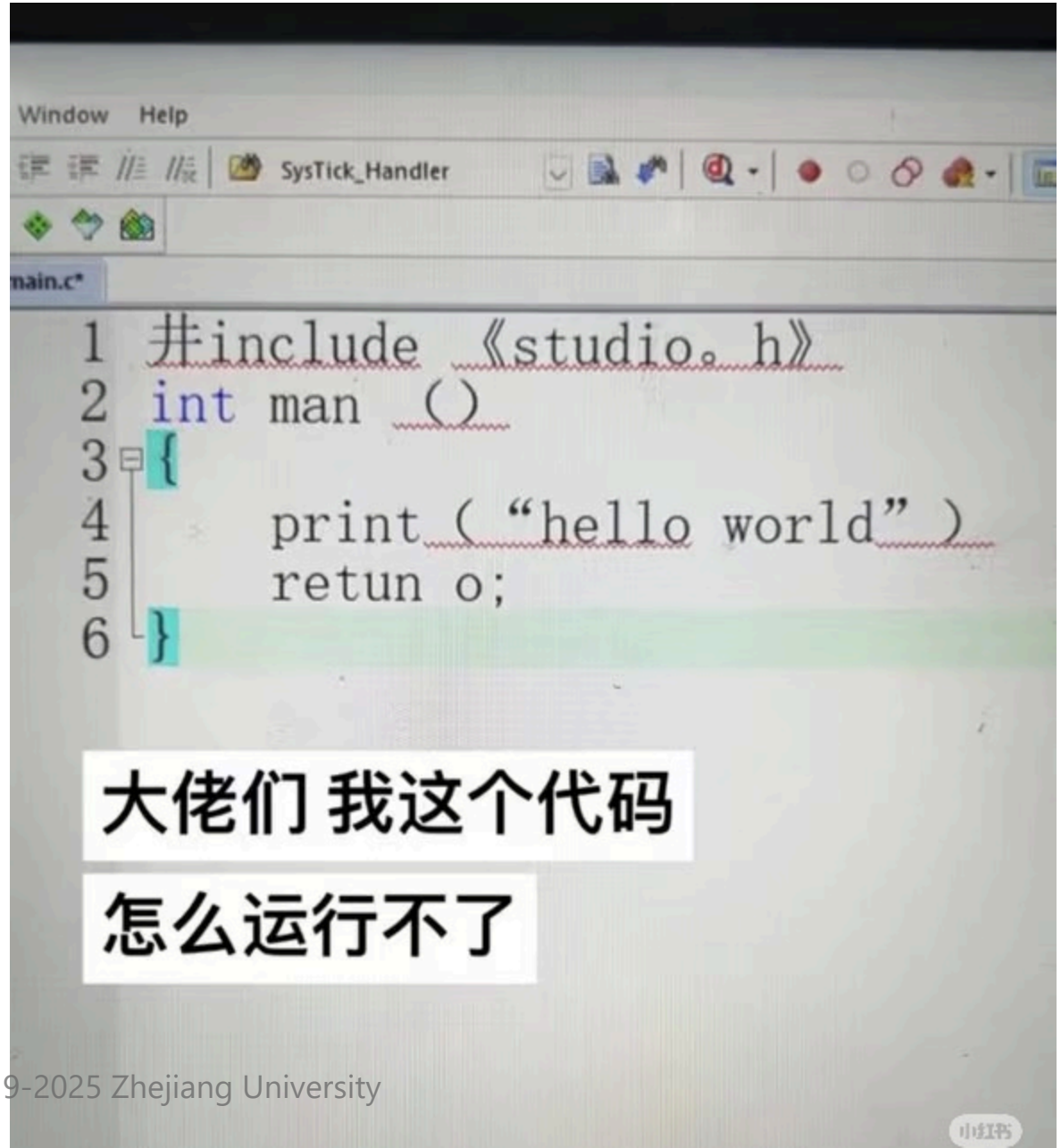
```c
#include <stdio.h>

int main()
{
   printf("Hello World")
}
```

- Red tilde means bug around it
- The specific reason lists in the terminal window
- 一定要仔细阅读错误信息的每一个单词

不要用中文

- 中国学生还有一个极其常见的低级错误，就是用了中文输入法来输入程序
- 那些标点符号，在中文和英文可能看上去相似，但是对于计算机是完全不同的符号
- 如果你还开了全角标点的话，问题就更严重了



Lab 1:

7-1: What is a computer?

Cube

IPO model

- Write a program reads an integer in and prints the cube of that integer
- Input — Process — Output

```c
#include <stdio.h>

int main()
{
    int x =0;
    int cube =0;
    //  input
    scanf("%d", &x);
    //  process
    cube = x*x*x;
    //  output
    printf("%d\n", cube);
}
```

IPO

- Input: to get and record data from user
  - Variable: name that presents the data
  - `scanf`: read(scan) data from the user's input
- Output: to show the result to user
  - `printf`: "print" the formated result
- Process: to do the calculation
  - operators and expression

Variable

- The facilities used for storing a program’s input data and its computational results are called variables because the values stored in variables can change (and usually do) as the program executes
- Variable is the place to hold data. Any time we need a place to hold data, we need a variable

变量是值的名字

- 用字母表示一个数，比如：
  - 用 a 表示小明的年龄，$a+2$ 就是两年后的年龄
  - 长方形的面积公式：$S = a \times b$，字母 a 和 b 代表的是具体的数（长、宽）

Definition of variable

- A variable has to be defined or declared before its first usage
- A variable can be named as one letter or one word, or any combination

Definition of variable

`<type> <name>;`

```
int price;
int amount;
int x;
int cube;
int price, amount;
```

变量没有缺省初始值

- 变量定义只是明确了有这样一个变量，但并没有给出它的初值
- 这样的变量中的初值是不确定的（不是随机的）
  - 有可能正好是`0`
- 作为好的编程习惯，应该在定义变量的时候给出确定的初值（即使很快就会重新赋值）

```
int x =0;
scanf("%d", &x);    //  x gets value by scanf
```

Definition of variable, again

`<type> <name>;`

```
int price = 0;
int amount = 0;
int x = 0;
int cube = 0;
int price = 0, amount = 0;
```

Pitfall

```
int price, amount = 0;
```

- In this case, `amount` gets an initial value `0`, while `price` does **NOT**

Input

```
scanf("%d", &x);
```

- Ask the function `scanf` to read and parse the next integer from user's input and set the value to `x`
- Be careful on the `&` punctuation
- Input your data in the terminal window
- The input is line by line. The indicator of the end of the line is the `enter` key. Before you press the `enter` key, nothing could be read by your program, and you still have chance to edit it

Output

```c
printf("hello World!\n");
```

- What inside the pair of `" "` is a string, which will be printed out by the `printf`
- `\n` means there will be a new line after this symbol, and `\n` itself will not be printed

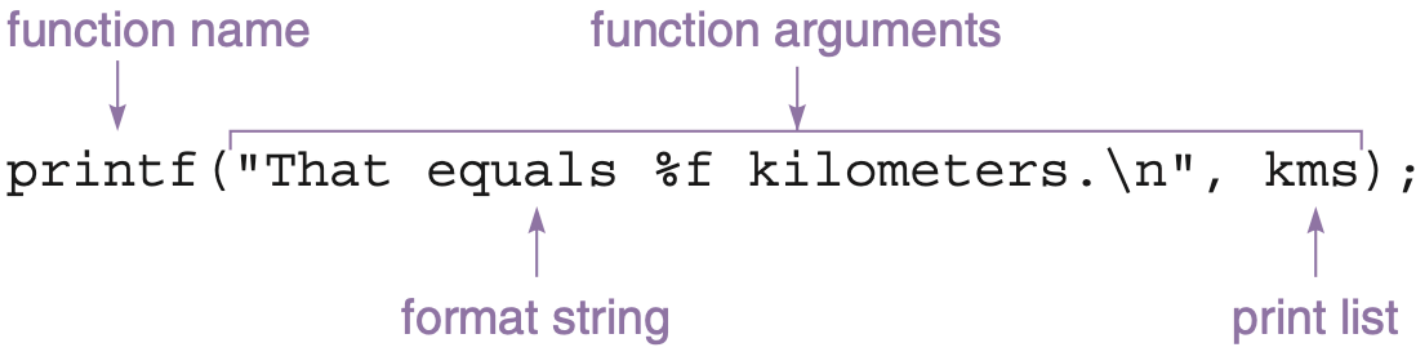
Output

- To see the results of a program's execution, we must have a way to specify what variable values should be displayed

```
printf("%d %d\n",  23+43, 22);
```

- `%d` means there is going to be an integer to be output at this place

```
printf("23+43=%d\n",  23+43);
```



Elementary arithmetic

arithmetic	C operator	meaning
$+$	`+`	addition
$-$	`-`	substraction
$\times$	`*`	multiplication
$\div$	`/`	division
	`%`	remainder

Assignment

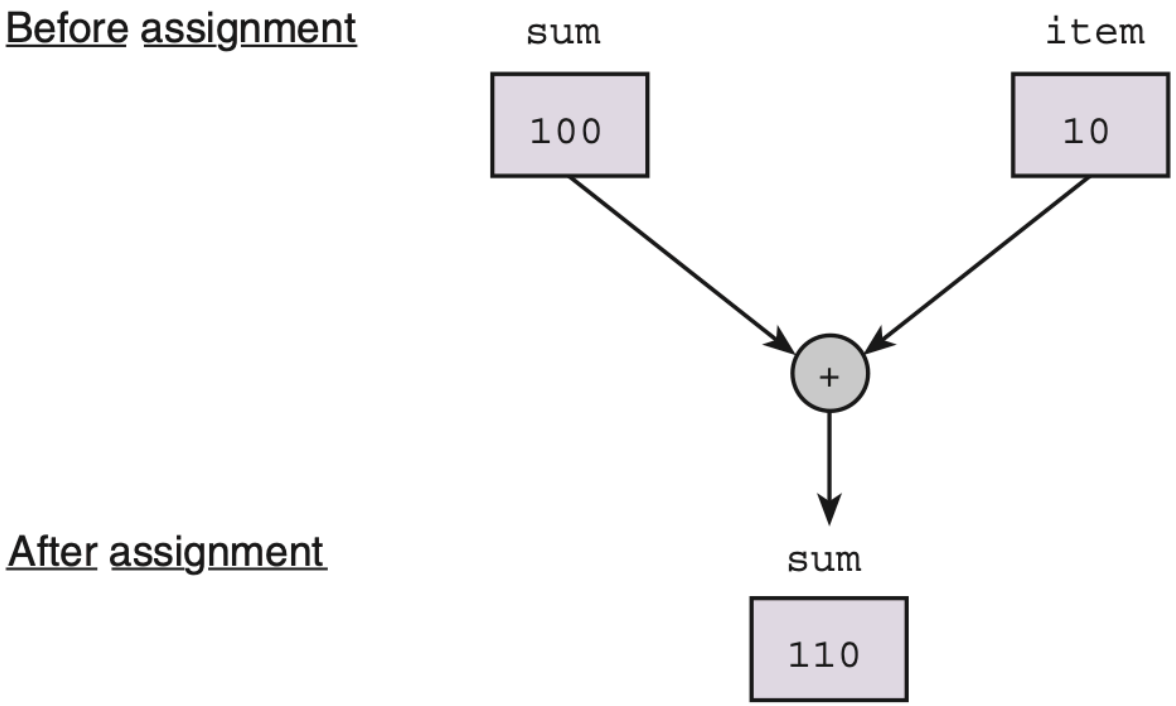
- An assignment statement stores a value or a computational result into a variable, and is used to perform most arithmetic operations in a program.

```
a = 6;
b = a;
```

- In arithmetic, $a = b$ means the relationship between a and b, i.e, a and b have the identical values.
- But here in programming, `a=b` means an action to be taken that make `a` the value of `b`
- The relationship is static, while action is dynamic. In arithmetic, $a = b$ and $b = a$ are two equal relationship. While in programing, they are totally the opposite

Understand this

```
sum = sum + item;
```



Lab 2: more input

7-2: a+b

```
scanf("%d %d", &a, &b);
```

Time difference

- The program reads four integers in, which represent two spots, and calculate the time difference between two spots

```
int hour1, hour2;
int minute1, minute2;

scanf("%d:%d", &hour1, &minute1);
scanf("%d:%d", &hour2, &minute2);
```

A strait way

```c
int hour1=0, hour2=0;
int minute1=0, minute2=0;

scanf("%d:%d", &hour1, &minute1);
scanf("%d:%d", &hour2, &minute2);

int hd = hour2 - hour1;
int md = minute2 - minute1;
printf("%d:%d\n", hd, md);
```

- What if `minute1` > `minute2`?

What if?

- If `minute1` > `minute2`, we have to borrow one from hour

```
int md = minute2 - minute1;
if ( minute1 > minute2 ) {
    md = 60+md;
    hd = hd-1;
}
printf("%d:%d\n", hd, md);
```

```c
int hour1=0, hour2=0;
int minute1=0, minute2=0;

scanf("%d:%d", &hour1, &minute1);
scanf("%d:%d", &hour2, &minute2);

int hd = hour2 - hour1;
int md = minute2 - minute1;
if ( md<0 ) {   //  if m<0 then...
    md = 60+md;
    hd = hd-1;
}
printf("%d:%d\n", hd, md);
```

Lab 3

5-1：时间差

`if`

```
if ( condition ) {
        …
}
```

- A compound statement, written as a group of statements bracketed by `{` and `}`, is used to specify sequential flow
- Control structures control the flow of execution in a program or function. The C control structures enable you to combine individual instructions into a single logical unit with one entry point and one exit point.

Relational operators

operator	description
`==`	equal to
`!=`	not equal to
`>`	greater than
`>=`	greater than or equal to
`<`	less than
`<=`	less than or equal to

- They are used to calculate relationship between two values

Result of relational operation

- `1` for expectation, while `0` for not

```
printf("%d\n", 5==3);
printf("%d\n", 5>3);
printf("%d\n", 5<=3);
```

Even?

- The statement below assigns the value 1 (true) to even (type int) if n is an even number:

```
even = (n % 2 == 0);
```

- Because all even numbers are divisible by 2,the remainder of n divided by 2 (`n % 2` in C) is 0 when n is an even number. The expression in parentheses compares the remainder to 0, so its value is `1` (true) when the remainder is 0 and its value is `0` (false) when the remainder is nonzero.

Precedence

- All relational operators are lower than all arithmetical ones, but higher than the assignment

```
7 >= 3 + 4
r = a>0;
```

Precedence

- Operators for equality, `==` and `!=`, are lower than other relational ones. For operators with same precedence, they should be calculated from left to right

```
5 > 3 == 6 > 4
6 > 5 > 4
a == b == 6
a == b > 0
```

Sum up

$$sum = \sum_{i=1}^{n} x_i$$

- Ask the user to enter a series of positive integers, and then enter `-1` to indicate the end of input. The program then calculates the sum of these numbers and outputs the number of numbers entered and the total sum.
- Think about:
  - variables -> algorithm -> (flow chart) -> program

Variables

- A variable that records the integers read
- How do you calculate the sum?
- Just add each number you read to a cumulative variable, and when all the data has been read, you will get the total sum and the number of items
- One variable (`sum`) records the cumulative result, and another variable (`cnt`) records the number of items read

Test case

- Make a series of numbers to be used as input
- Calculate the sum by hand first
- May design more than one case

IPO model

```c
#include <stdio.h>

int main()
{
    int sum = 0;
    int cnt = 0;

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

- input—>variable—>output

One number first

- First, consider how to read a number

```
int x;
scanf("%d", &x);
sum += x;
cnt ++;
```

- From this, we can see that previously, it was necessary to:

```
int sum =0;
int cnt =0;
```

For the second number

```
int x;
scanf("%d", &x);
sum += x;
cnt ++;
scanf("%d", &x);
sum += x;
cnt ++;
```

Make a loop

- Find the tasks that need to be repeated and wrap them in a `while(1)` loop

```c
int sum = 0;
int cnt = 0;

int x;
while (1) {
    scanf("%d", &x);
    sum += x;
    cnt ++;
}
printf("%d, %d\n", sum, cnt);
```

- Make a `while(1)` around the “reading one number”

Leave that loop

- Find the when (where) and why (a reason) to leave the loop

```c
int sum = 0;
int cnt = 0;

int x;
while (1) {
    scanf("%d", &x);
    if ( x== -1 ) {
        break;
    }
    sum += x;
    cnt ++;
}
printf("%d, %d\n", sum, cnt);
```

- Put a `break` in the clause that `x` is `-1`

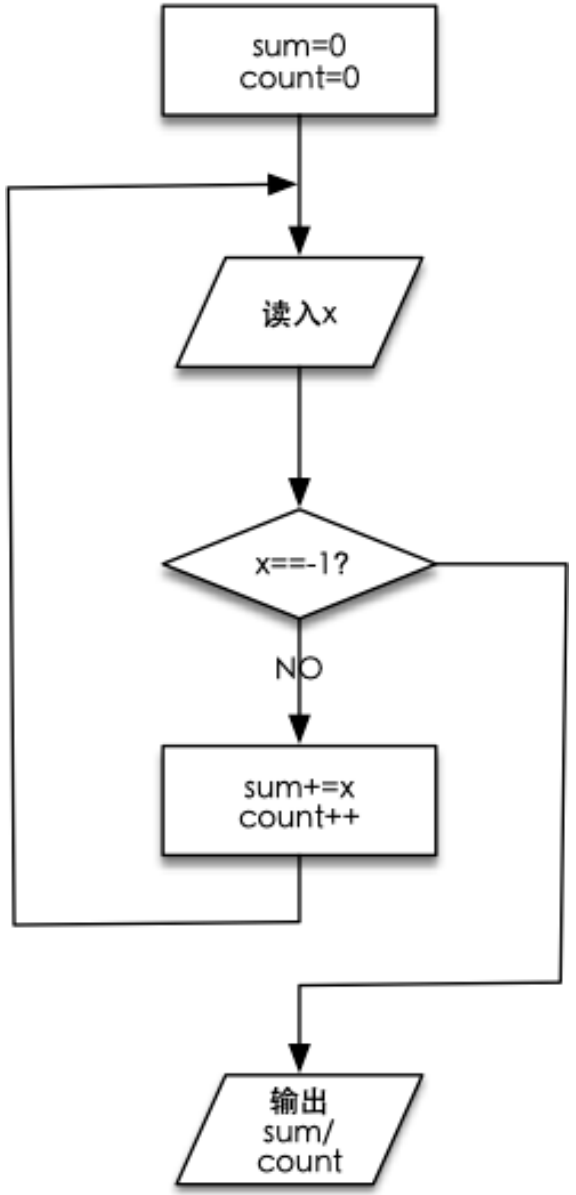
Break a dummy loop

- `break` is to exit the loop

```
while (1) {
    …
    if (…) {
        break;
    }
}
```

Algorithm

1. Initialize variables `sum` and `cnt` to 0;
2. Read in `x`;
3. If `x` is `-1`, end the loop;
4. Add `x` to `sum`, increment `cnt` by 1, and return to step 2;
5. Print `sum` and `cnt`.



Lab 4

5-2: Sum Up

What we've learned today?

- The basic skeleton of a C program
- IPO model
- Input and output
- Variables
- Elementary arithmetic
- Relational operations
- `if` statement
- `while (1)` statement with `break`