C程序设计基础

Introduction to C programming

Lecture 3: Basics

张振国 zhangzg@sustech.edu.cn

南方科技大学/理学院/地球与空间科学系

Review on L2

一个程序应包括两个方面的内容:

- 对数据的描述：数据结构(data structure)
- 对操作的描述：算法(algorithm)

著名计算机科学家沃思提出一个公式:

数据结构 + 算法 = 程序

完整的程序设计应该是:

数据结构＋算法＋程序设计方法＋语言工具

算法的表示

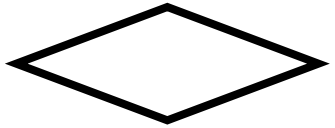
可以用不同的方法表示算法，常用的有：

- 自然语言
- 传统流程图
- 结构化流程图
- 伪代码
- PAD图

流程图

美国国家标准化协会ANSI(American National Standard Institute)规定了一些常用的流程图符号：

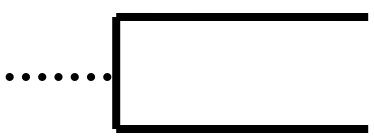
起止框



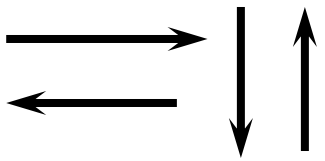
判断框

处理框

输入/输出框



注释框



流向线

连接点

流程图

例2.9 将例2.4的算法用流程图表示

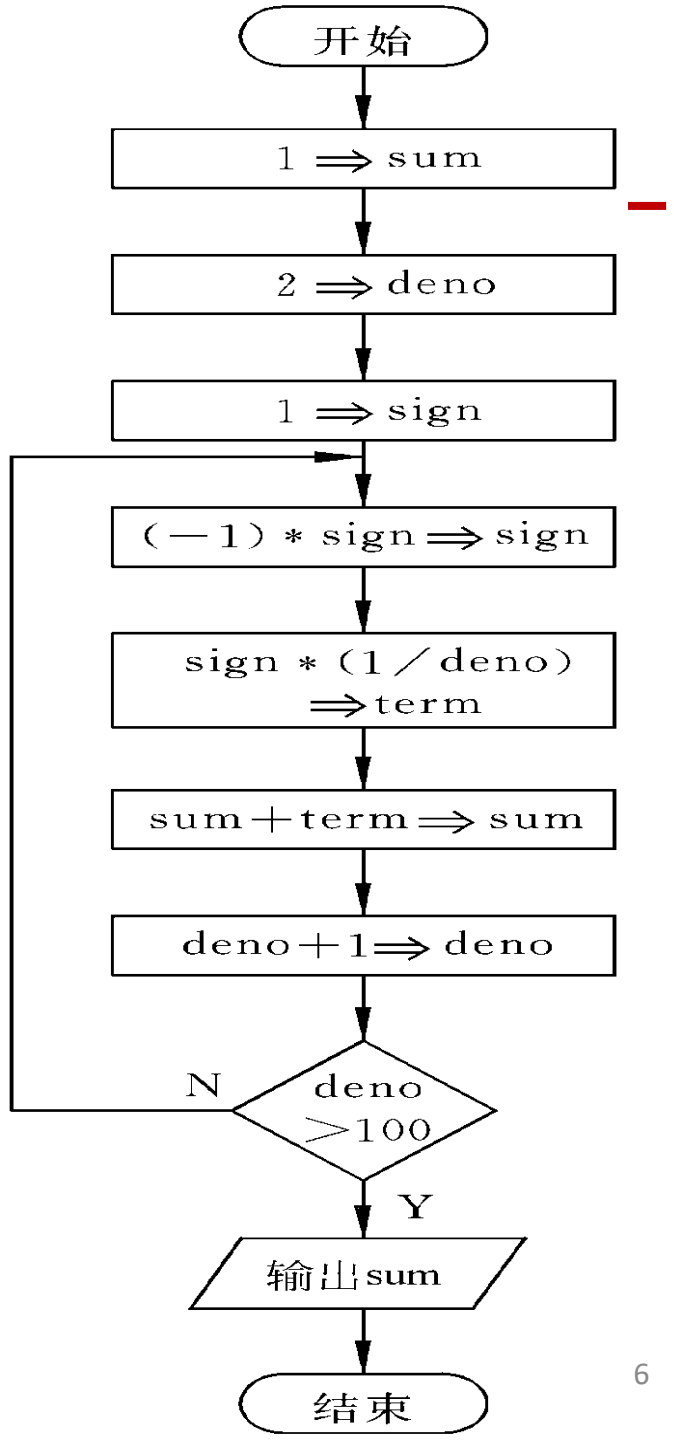
$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots-\frac{1}{99}+\frac{1}{100}$$

- 流程图是表示算法的较好的工具。一个流程图包括以下几部分 ：

(1) 表示相应操作的框；

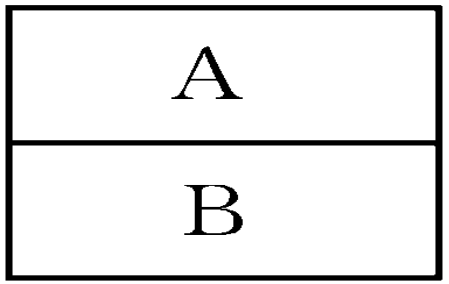
(2) 带箭头的流程线；

(3) 框内外必要的文字说明。

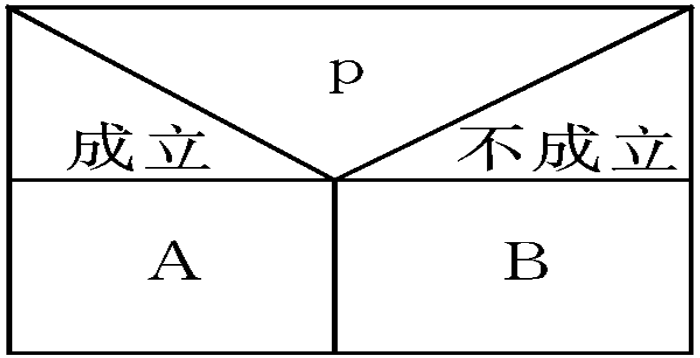


N-S图

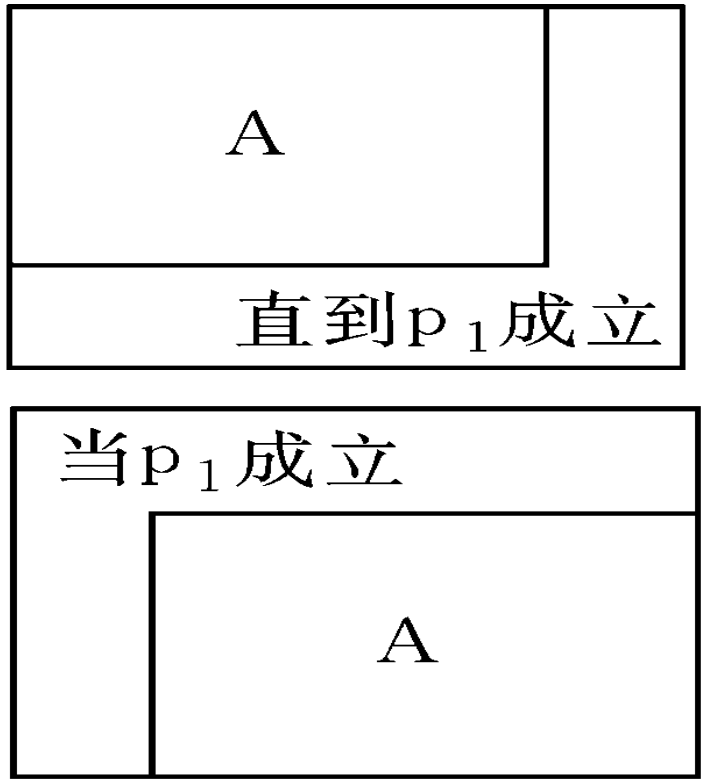
N-S流程图用以下的流程图符号：



(1)顺序结构

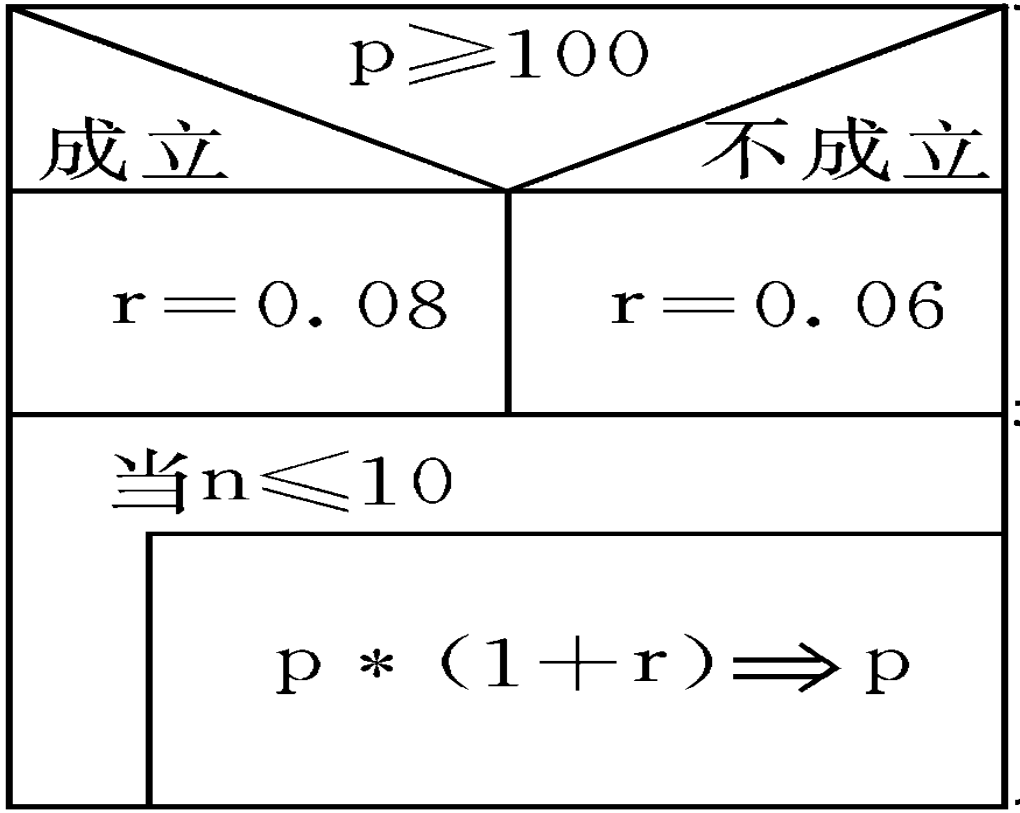


(2)选择结构



(3)循环结构

◆用三种N-S流程图中的基本框，可以组成复杂的N-S流程图。图中的A框或B框，可以是一个简单的操作，也可以是三个基本结构之一。



A

A框可以是一个选择结构

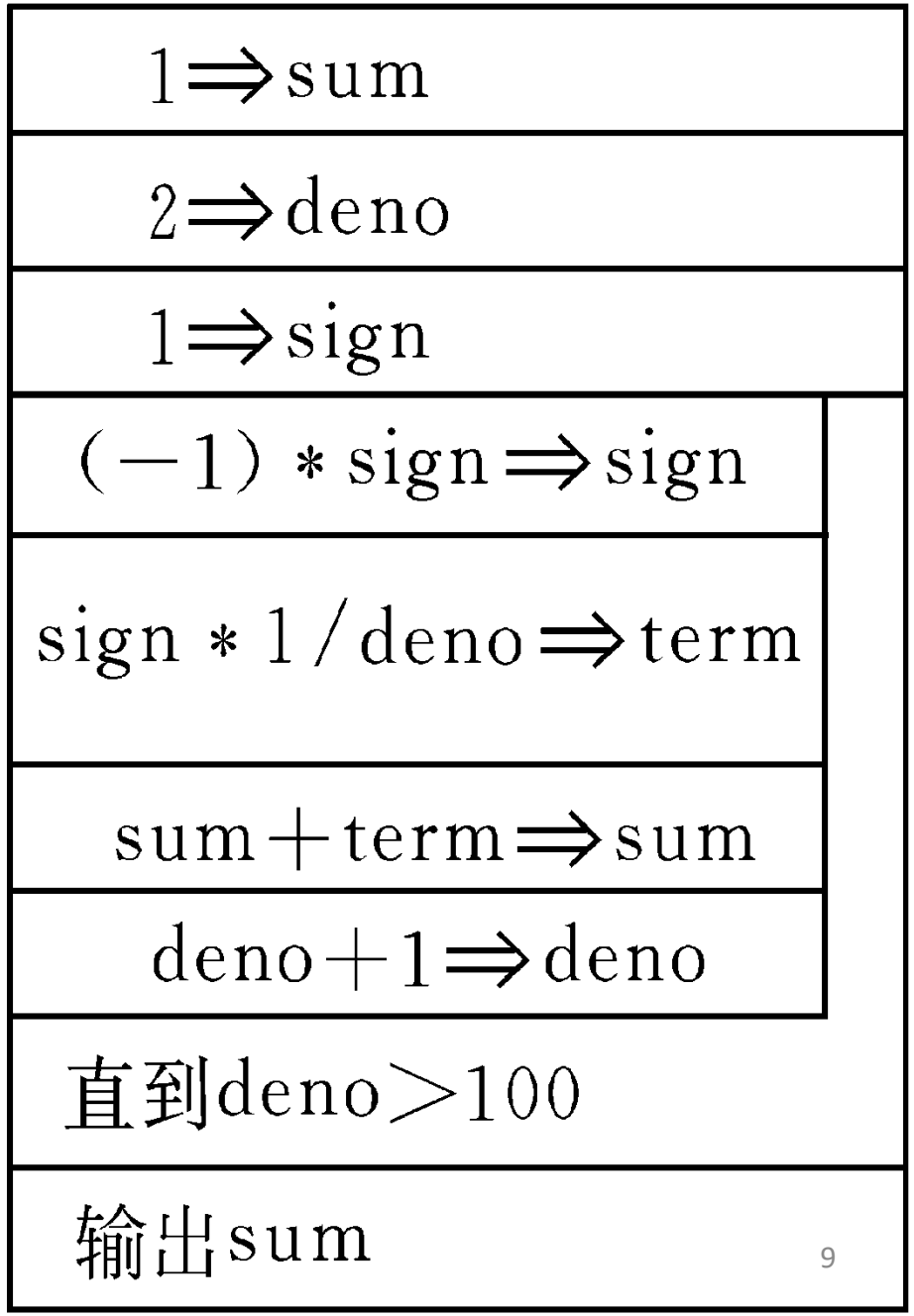
B

B框可以是一个循环结构

N-S图

例2.14 将例2.4的算法用N-S图表示

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots-\frac{1}{99}+\frac{1}{100}$$



伪代码

例2.17 输出50个学生中成绩高于80分者的学号和成绩。

用伪代码表示算法：

```
BEGIN｛算法开始｝
    1 → i
    while i≤50
      {input ni and gi
       i+1 → i}
    1 → i
  while i≤50
   {if gi≥80 print ni and gi
  i+1 → i}
   END｛算法结束｝
```

2.5 结构化程序设计方法（上节课）

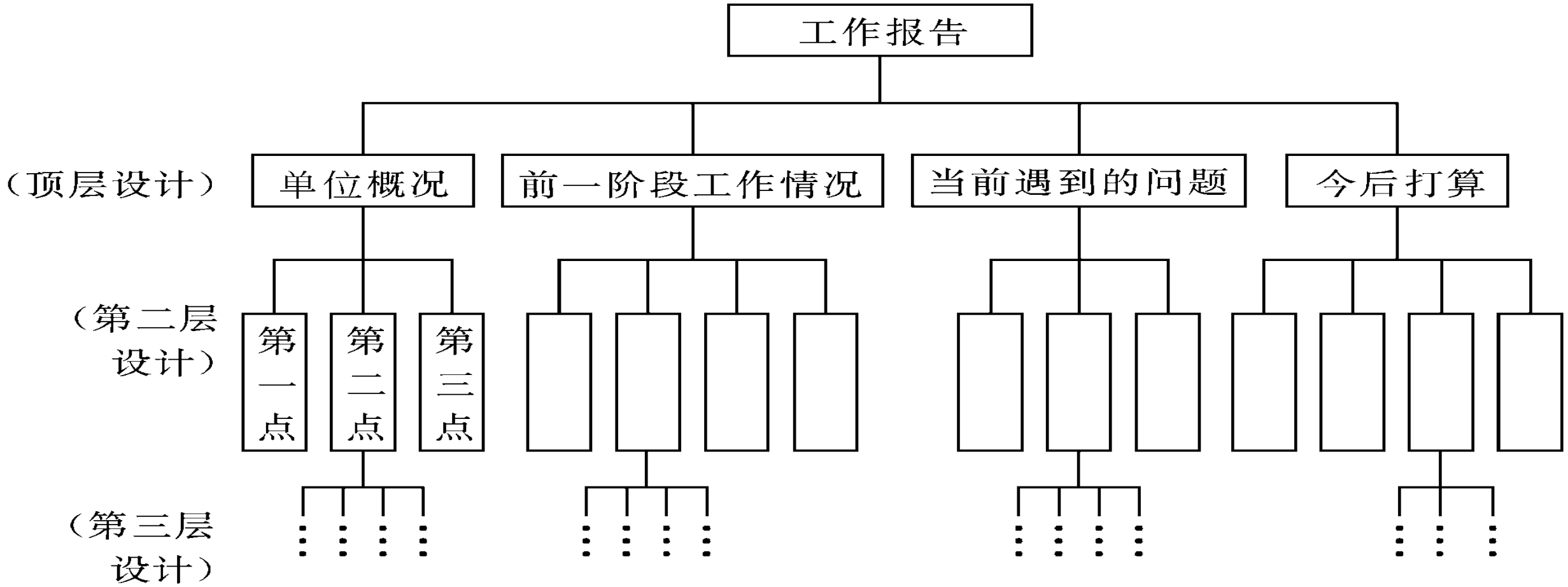
- 一个结构化程序就是用高级语言表示的结构化算法。用三种基本结构组成的程序必然是结构化的程序，这种程序便于编写、便于阅读、便于修改和维护。
- 结构化程序设计强调程序设计风格和程序结构的规范化，提倡清晰的结构。
- 结构化程序设计方法的基本思路是：把一个复杂问题的求解过程分阶段进行，每个阶段处理的问题都控制在人们容易理解和处理的范围内。

采取以下方法来保证得到结构化的程序：

- 自顶向下；
- 逐步细化；
- 模块化设计；
- 结构化编码。

两种不同的方法：

- 自顶向下，逐步细化；
- 自下而上，逐步积累。



用这种方法逐步分解，直到作者认为可以直接将各小段表达为文字语句为止。这种方法就叫 做“自顶向下，逐步细化”。

自顶向下，逐步细化方法的优点：

考虑周全，结构清晰，层次分明，作者容易写，读者容易看。如果发现某一部分中有一段内容不妥，需要修改，只需找出该部分修改有关段落即可，与其它部分无关。我们提倡用这种方法设计程序。这就是用工程的方法设计程序。

模块设计的方法：

- 模块化设计的思想实际上是一种“分而治之”的思想，把一个大任务分为若干个子任务，每一个子任务就相对简单了。
- 在拿到一个程序模块以后，根据程序模块的功能将它划分为若干个子模块，如果这些子模块的规模还嫌大，还再可以划分为更小的模块。这个过程采用自顶向下方法来实现。
- 子模块一般不超过50行。
- 划分子模块时应注意模块的独立性，即：使一个模块完成一项功能，耦合性愈少愈好。

总结（上节课）

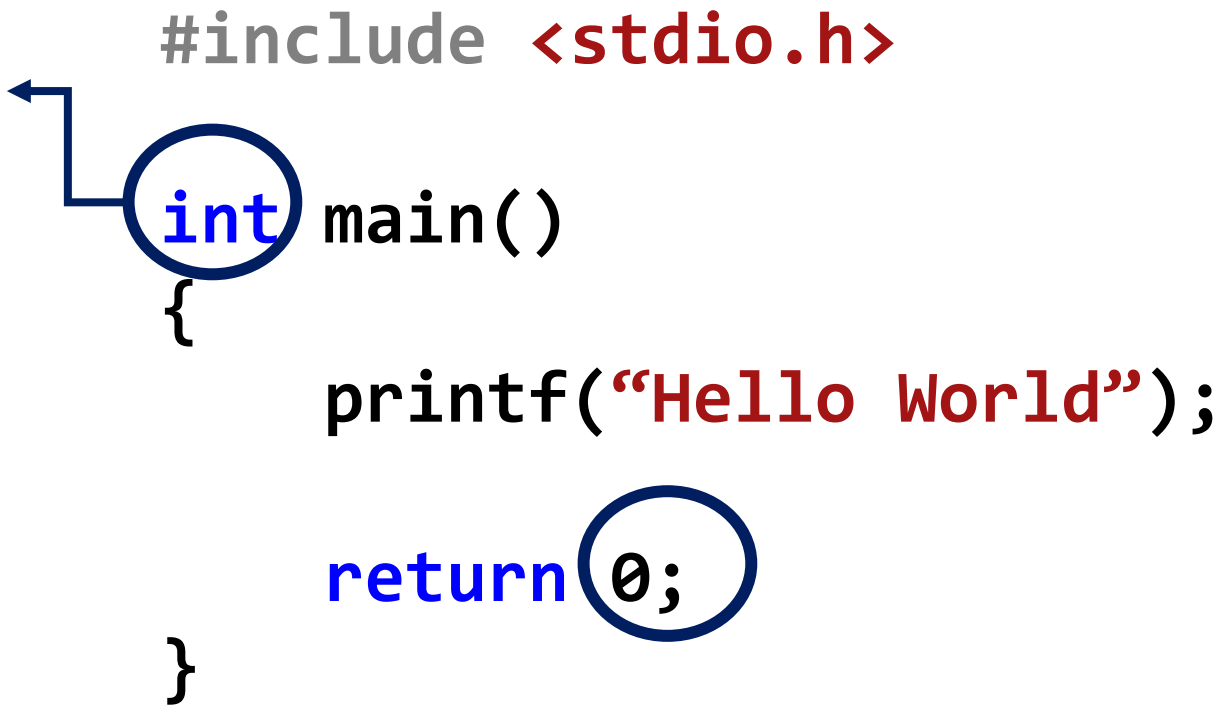
- 工欲善其事，必先利其器
- 至少熟练一种算法流程图，不必贪多求全

Content

- **Data types and variables**
- **Operations and expressions**
- **Formatted Input/Output**

Data types and variables

You may still remember HelloWorld example?



`int` is a data type, ask the program to return an integer number!

Data types and variables

```
int num = 5;      //整数
float x = 2.14; //浮点数、实数
char  c = ‘T’;  //字符串
char  s[10] = “Hello”;  //字符串
```

Data types

Original K&R Keywords	C90 K&R Keywords	C99 Keywords
int	signed	_Bool
long	void	_Complex
short		_Imaginary
unsigned		
char		
float		
double		

Variables

Variables are placeholders for values, each variable has a **type** defined. The type determines how it is stored and how much space (bit) it needs in machine.

```
type variable;                /*declare*/
type variable = value; /*initialize*/
```

int num; //声明
num = 5; //赋值
printf(“num = %d”, num);

int num = 5; //声明+赋值
printf(“num = %d”, num);

Variables

A variable name can ONLY be defined once, but its value can be set multiple times!

```
int num = 5; //声明+赋值
printf("num = %d", num);

num = 10; //重新赋值
printf("num = %d", num);
```

```
int num = 5; //声明+赋值
printf("num = %d", num);

int num = 10; //重定义
printf("num = %d", num);
```

Variables

Declare and initialize a variable separately

```
int a, b, c;
a = 3;
b = 4;
c = 100;
```

Declare and initialize a variable jointly

```
int a = 3, b=4, c=100;
float f=3.14;
double d = 1.2321232;
char c ='A';
```

Variables

Constant variable（常量）

```
const int x = 3; //声明+赋值
int y = 5;

y = 10; //重新赋值
x = 6;
```

Type casting（类型转换）

```
float x;
int y = 5;

x = (float)y;
```

Rules to name variables?

- Lowercase/uppercase letters, digits and the underscore(_)
- The first character must not be a number.
- Length limit (<=31)
- Case-sensitive

Valid Names	Invalid Names
wiggles	$Z]**
cat2	2cat
Hot_Tub	Hot-Tub
TaxRate	tax rate
_kcab	don’t

Rules to name variables?

- Keywords are reserved by C, cannot be used

No need to memorize keywords, IDE will warn you! However…

ISO C Keywords

auto	extern	short	while
break	float	signed	_Alignas
case	for	sizeof	_Alignof
char	goto	static	_Bool
const	if	struct	_Complex
continue	*inline*	switch	_Generic
default	int	typedef	_Imaginary
do	long	union	_Noreturn
double	register	unsigned	_Static_assert
else	restrict	void	#_Thread_local
enum	return	volatile	

Rules to name variables?

- Keywords are reserved by C, cannot be used
- Variable names must be **unique**
- Variable names should be **readable**, **meaningful** and **consistent**

  - UpperCamelCase - BodyMassIndex
  - lowerCamelCase - bodyMassIndex
  - snake_case – body_mass_index

Rules to name variables?

Good names:

```
int face_num;
int numOfDetectedFaces;
int DetFaceNum;
```

Prohibited names:

```
int float;
int main;
int return;
```

Bad names:

```
int test1, test2, test3;     //not meaningful
int jack, marry;  //hard to understand
int face_num, BodyMassIndx; //style not consistent
```

Variables

Example: create a variable list for students

```
#include <stdio.h>

int main ()
{
    char name[6] = "Tom";
    char gender = 'M';
    int age = 18;
    float height = 1.78;
    int grade = 88;
   return 0;
}
```

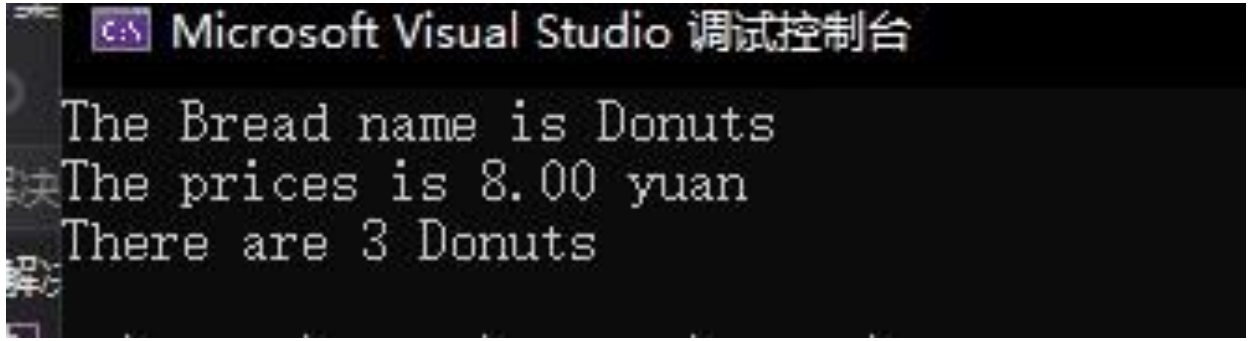
```
Microsoft Visual Studio 调试控制台
name = Tom
gender = M
age = 18
height =1.780000 m
grade = 88
```

Variables

Example: create a variable list for food

```
#include <stdio.h>

int main ()
{
    char name[10] = "Donuts";
    float price = 8.0;
    int num = 3;
    return 0;
}
```



Variables

Example: create a variable list for animals

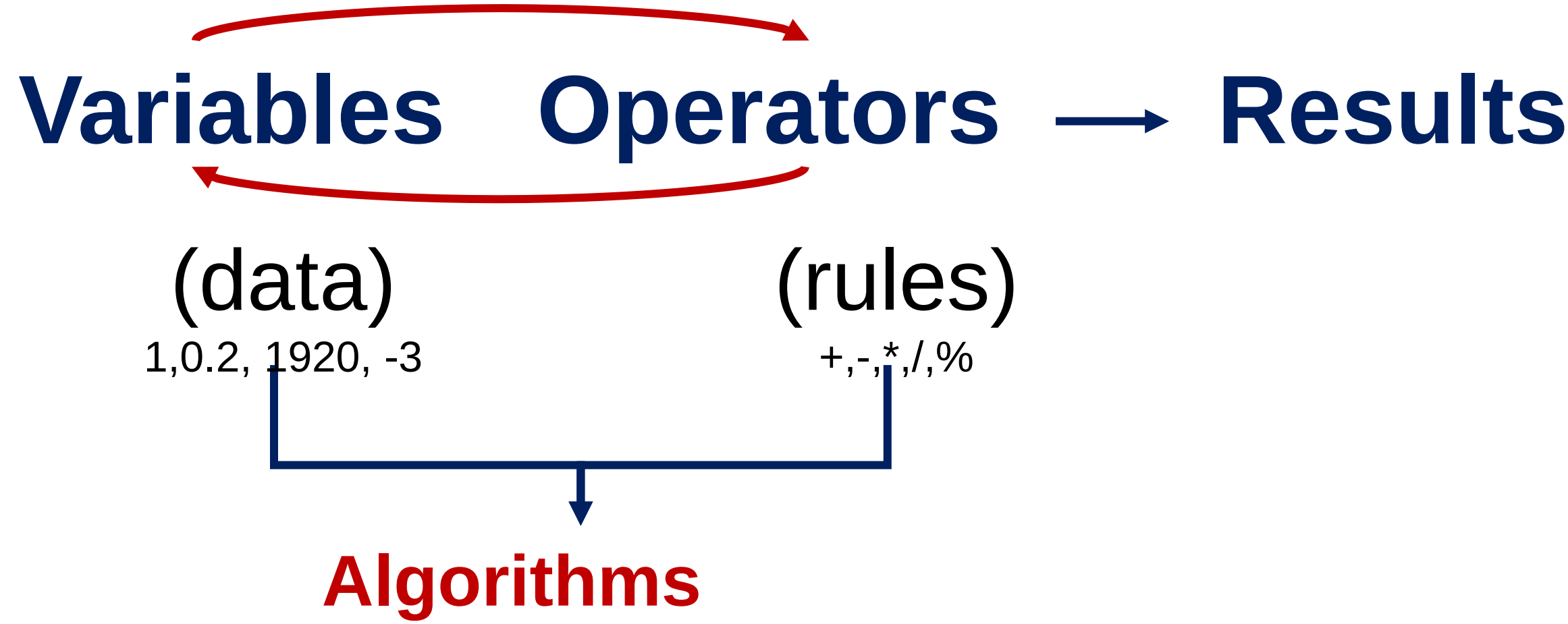
```
#include <stdio.h>

int main ()
{   char animal[10] = "Elephant";
    char name[5] = "Elly";
    char gender = 'F';
    int age = 3;
    float weight = 2.03;
    return 0;
}
```

Content

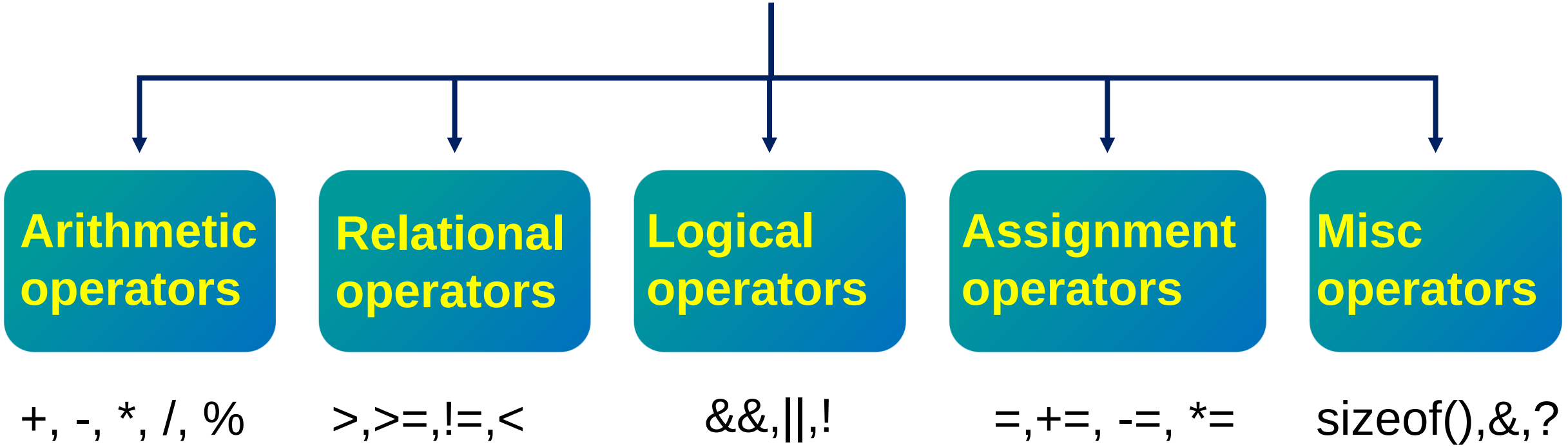
- Data types and variables
- Operations and expressions
- Formatted Input/Output

Operations


Variables
Operators
Results
(data)
1,0.2, 1920, -3
(rules)
+,-,*,/,%
Algorithms

Operators

Operator is a symbol that tells compiler to perform specific mathematical or logical operations.



Arithmetic Operators

Define two variables: `int A = 5, B = 3;`

Operators	Description	Example
+	Add two variables	A + B = 8
-	Subtract two variables	A – B = 2
*	Multiply two variables	A * B = 15
/	Divide two variables	A / B = 1
%	Take the reminder (only for `int`!)	A % B = 2
++	Increment by adding 1	A++ = 6
--	Decrement by subtracting 1	A-- = 4

Arithmetic Operators

More examples on different data types

Operators	int A = 10, B = 20;	float A = 13, B = 6;
+	A + B = 30	A + B = 19
-	A – B = -10	A – B = 7
*	A * B = 200	A * B = 78
/	A / B = 0	A / B = 2.166667
%	A % B = 10	A % B = ? (wrong!)
++	A++ = 11	A++ = 14
--	A-- = 9	A-- = 12

Arithmetic Operators

Post-increment A++
use A; then increment it

```
int A = 20;
int B = A++;

printf("A=%d\n",A);
printf("B=%d\n",B);
```

```
A=21
B=20
```

Pre-increment ++A
increment it; then use it

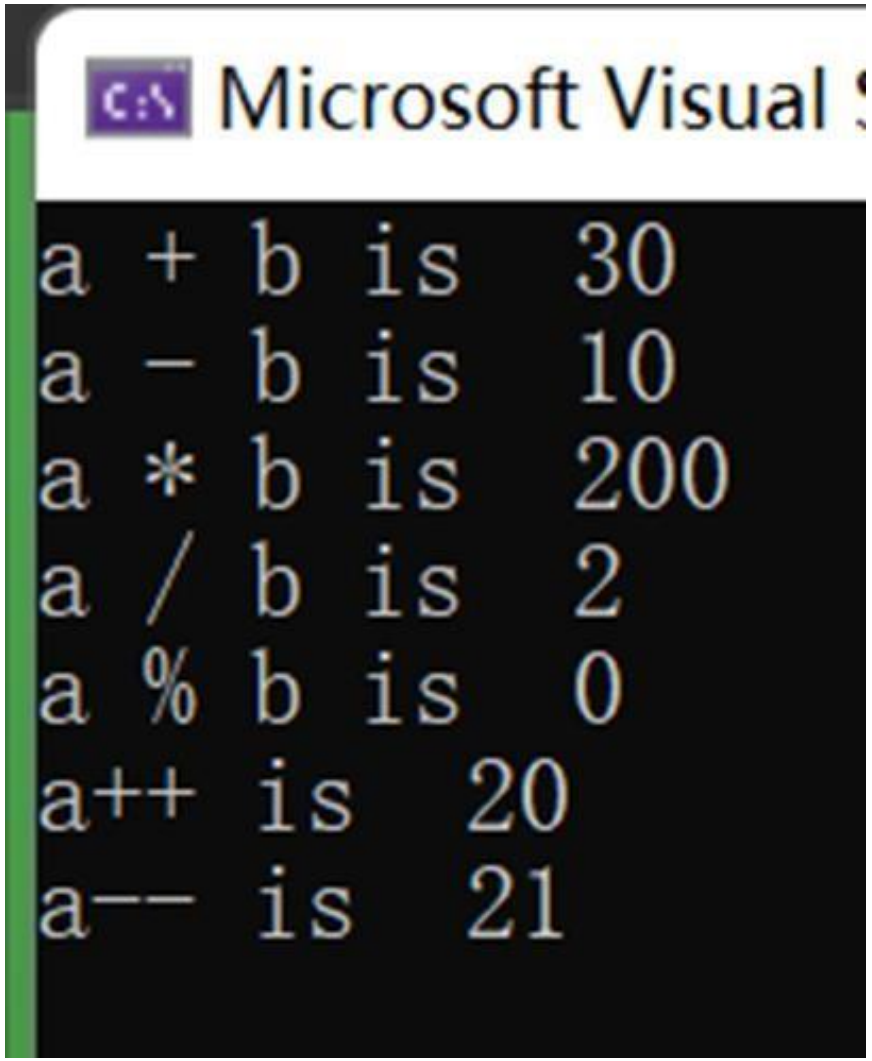
```
int A = 20;
int B = ++A;

printf("A=%d\n",A);
printf("B=%d\n",B);
```

```
A=21
B=21
```

Arithmetic Operators

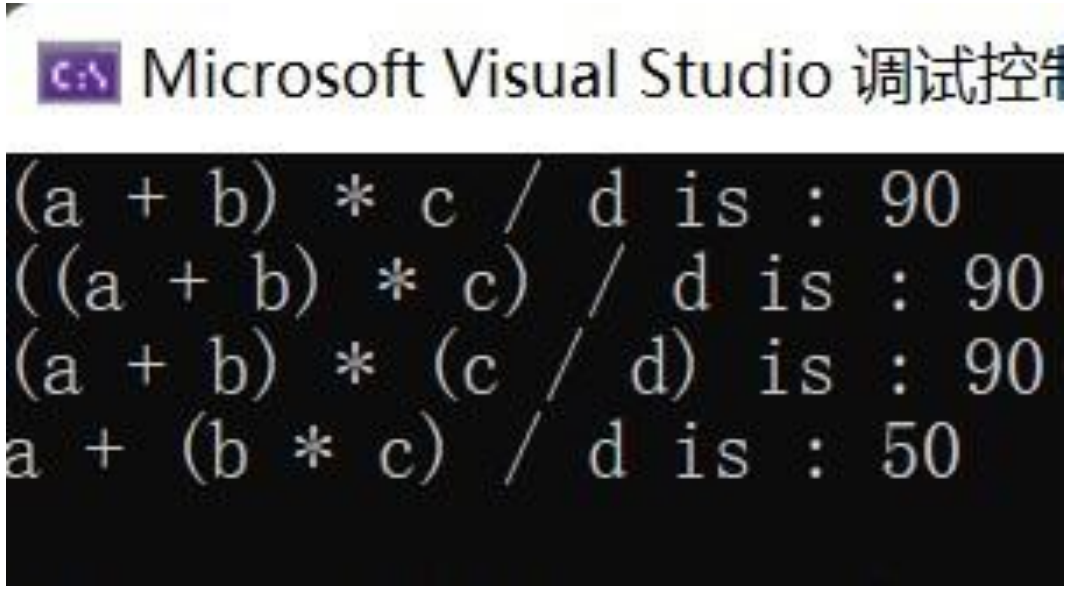
```
#include<stdio.h>
main(){
  int a = 20, b = 10;
  int c;
  c = a + b;
  printf("a + b is %d\n", c);
  c = a - b;
  printf("a - b is %d\n", c);
  c = a * b;
  printf("a * b is %d\n", c);
  c = a / b;
  printf("a / b is %d\n", c);
  c = a++;
  printf("a++  is %d\n", c);
  c = a--;
  printf("a--  is %d\n", c);
  return 0;
}
```



Arithmetic Operators

```
#include<stdio.h>
main(){
  int a = 20;
  int b = 10;
  int c = 15;
  int d = 5;
  int e;

  e = (a + b) * c / d;
  printf("(a + b) * c / d is : %d\n", e );
  e = ((a + b) * c) / d;
  printf("((a + b) * c) / d is : %d\n" , e );
  e = (a + b) * (c / d);
  printf("(a + b) * (c / d) is : %d\n", e );
  e = a + (b * c) / d;
  printf("a + (b * c) / d is : %d\n" , e );
}
```



Arithmetic Operators

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

  shoe = 17.0;
  while (++shoe < 18.5)
  {
    printf("The first Size is: %f\n", shoe);
  }




  return 0;
}
```

++

```
The first Size is:18.000000
```

Arithmetic Operators

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

++

```
  shoe = 17.0;
  while (shoe++ < 18.5)
  {
    printf("The second Size is: %f\n", shoe);
  }

  return 0;
}
```

```
The second Size is:18.000000
The second Size is:19.000000
```

Arithmetic Operators

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

  shoe = 17.0;
  while (++shoe < 18.5)
  {
    printf("The first Size is: %f\n", shoe);
  }
  shoe = 17.0;
  while (shoe++ < 18.5)
  {
    printf("The second Size is: %f\n", shoe);
  }

  return 0;
}
```

++

```
The first Size is:18.000000
```

```
The second Size is:18.000000
The second Size is:19.000000
```

Arithmetic Operators

++/--

Don’t use increment/decrement on a variable that

- is part of more than one argument of a function;
- appears more than once in an expression.

```
while (num < 21)
  {
    printf("%d %d\n", num, num*num++);
  }
```

```
ans = num/2 + 5*(1 + num++);
```

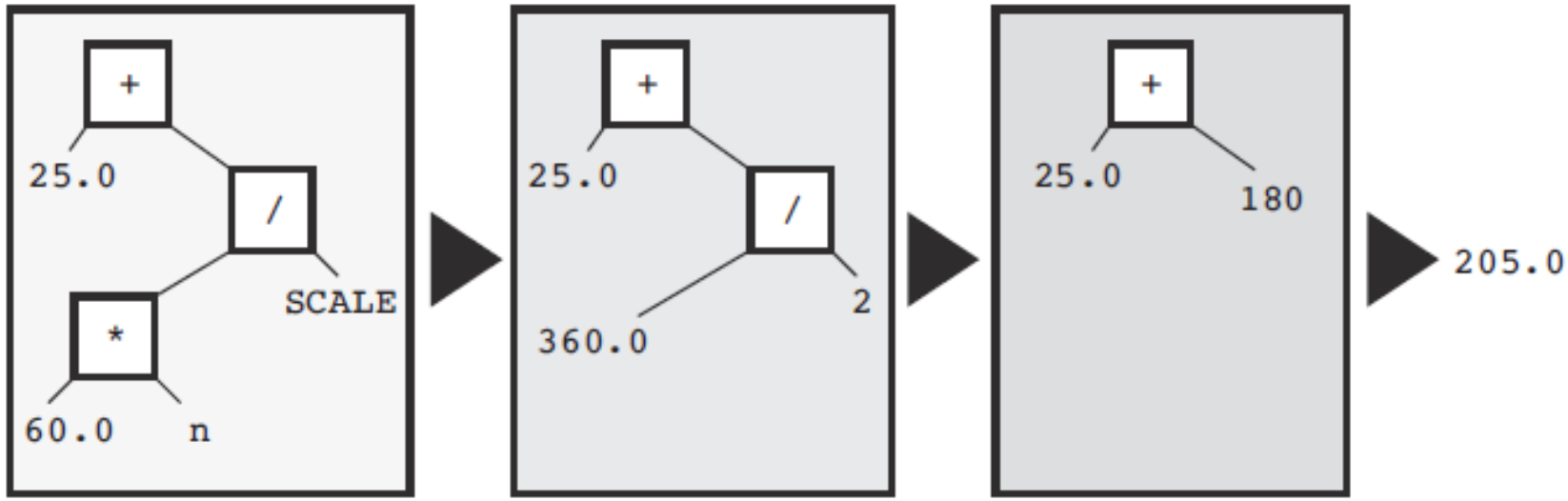
- 不要写出别人看不懂的也不知道系统会怎样执行程序

```
n = 3;
y = n++ + n++;
```

Operator precedence(优先级)

```
SCALE=2;
n=6;
butter=25.0+60.0*n/SCALE;
```

```
SCALE =2;
n=6;
butter=25.0+60.0*n/ SCALE;
```



Operator precedence(优先级)

Table 5.1 Operators in Order of Decreasing Precedence

Operators	Associativity
( )	Left to right
+ – (unary)	Right to left
* /	Left to right
+ – (binary)	Left to right
=	Right to left

结合性/结合律

++/--

unary(一元)

binary(二元)

```
float a;
a=+4.3/-5.0+-6.6;
```

It's allowed, but not recommended.

```
a = b += c++ -d + --e / -f;
```

```
(a = (b += (((c++) –d) + ((--e) / (-f))));
```

Precedence and the order of evaluation

```
int y;
y = 6*12 + 5*20;
```

(1) (2)

(1) and (2), which one will be evaluated first? It depends on hardware.

```
int y;
y = 12 / 3 * 2;
```

From left to right.
use () when it is needed.

Precedence and the order of evaluation

Operators with different characters, don’t be confused.

```
int i, j;

i+++j

(i++)+j

i+(++j)
```

Relational Operators

Define two variables: `int A = 5, B = 3;`

Operators	Description	Example
==	**Check if two variables are equal**	A==B = 0 (false)
!=	**Check if two variables are unequal**	A != B = 1 (true)
>	**Check if A is larger than B**	A > B = 1 (true)
<	**Check if A is smaller than B**	A < B = 0 (false)
>=	**Check if A is larger or equal than B**	A >= B = 1 (true)
<=	**Check if A is smaller or equal than B**	A <= B = 0 (false)

Relational Operators

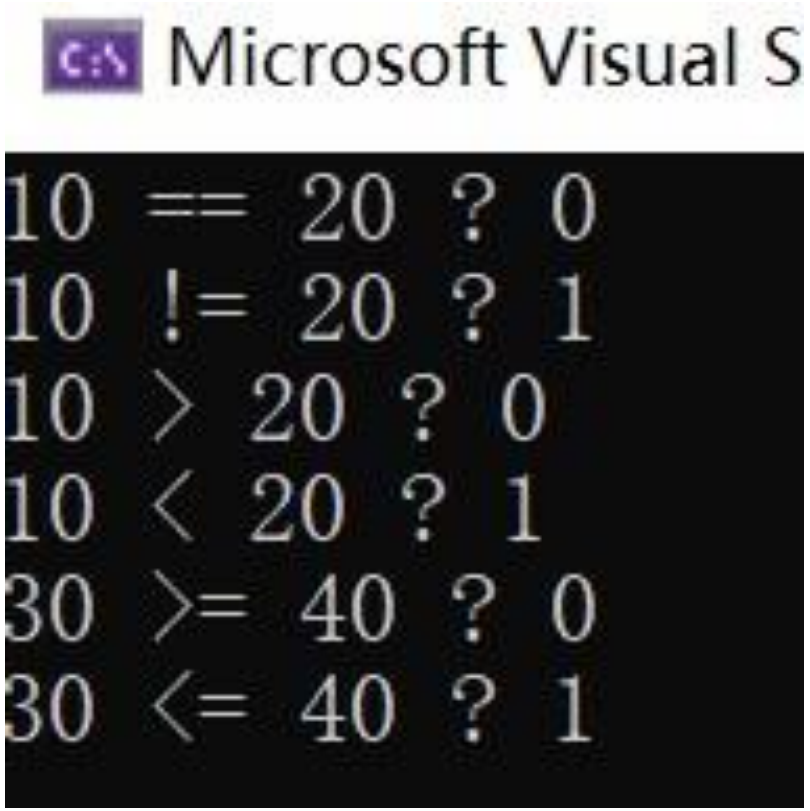
More examples on different data types

Operators	float A = 3.5, B = 3.5;	char A = ‘A’, B = ‘B’;
==	A==B = 1 (true)	A==B = 0 (false)
!=	A != B = 0 (false)	A != B = 1 (true)
>	A > B = 0 (false)	A > B = 0 (false)
<	A < B = 0 (false)	A < B = 1 (true)
>=	A >= B = 1 (true)	A >= B = 0 (false)
<=	A <= B = 1 (true)	A <= B = 0 (true)

Relational Operators

Example 1: comparing integers

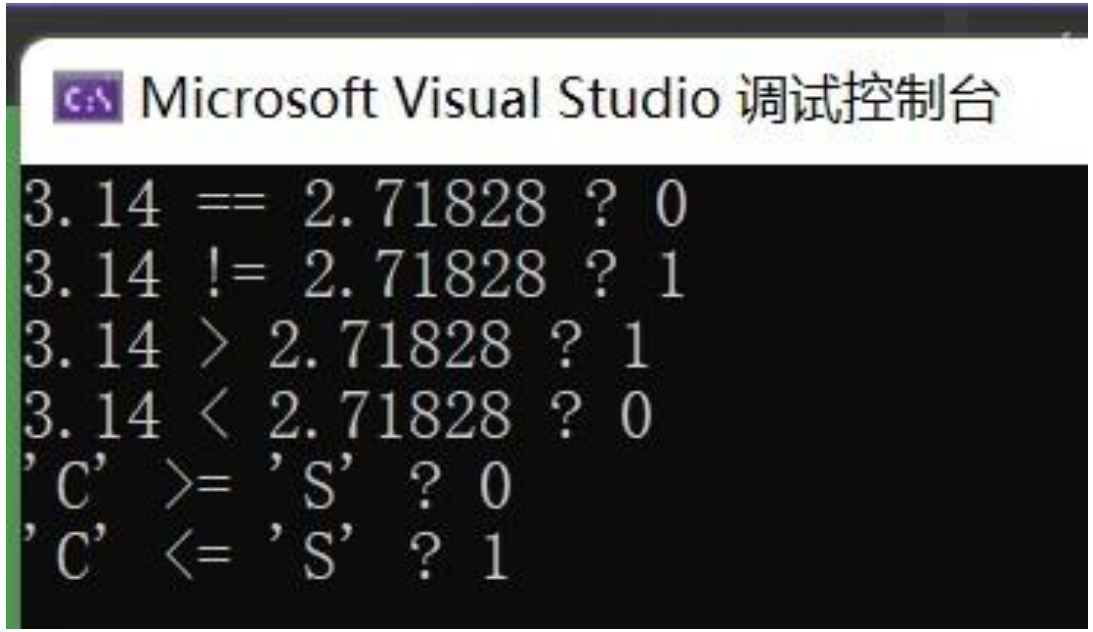
```
#include <stdio.h>
main()
{
        int a = 10;
        int b = 20;
        int c = 30;
        int d = 40;
        int e;
        e = a == b;
        printf("10 == 20 ? %d\n",e);
        e = a != b;
        printf("10 != 20 ? %d\n",e);
        e = a > b;
        printf("10 > 20 ? %d\n",e);
        e = a < b;
        printf("10 < 20 ? %d\n",e);
        e = c >= d;
        printf("30 >= 40 ? %d\n",e);
        e = c <= d;
        printf("30 <= 40 ? %d\n",e);
}
```



Relational Operators

Example 2: comparing floats or characters

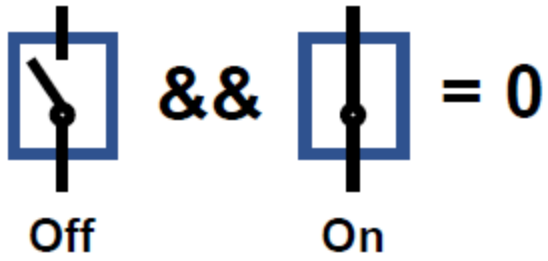
```
#include <stdio.h>
main()
{
        float a = 3.14;
        float b = 2.71828;
        char c = 'C';
        char d = 'S';
        int e;
        e = a == b;
        printf("3.14 == 2.71828 ? %d\n",e);
        e = a != b;
        printf("3.14 != 2.71828 ? %d\n",e);
        e = a > b;
        printf("3.14 > 2.71828 ? %d\n",e);
        e = a < b;
        printf("3.14 < 2.71828 ? %d\n",e);
        e = c >= d;
        printf("'C' >= 'S' ? %d\n",e);
        e = c <= d;
        printf("'C' <= 'S' ? %d\n",e);
}
```

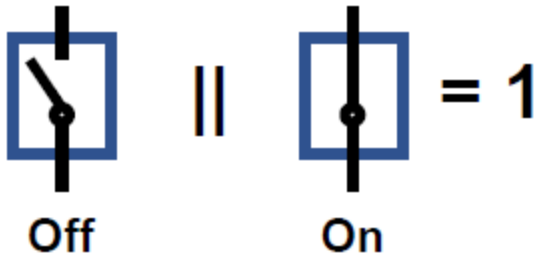


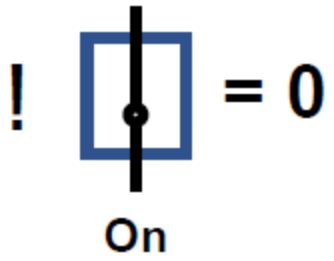
Logical Operators

Define two variables: `int A = 0, B = 1;`

Operators	Description	Example
&&	**AND operator, if both are on, then on**	A&&B = 0 (false)
||	**OR operator, if any is on, then on**	A ||B = 1 (true)
!	**NOT operator, turn opposite**	!A = 1 (true) !B = 0 (false)







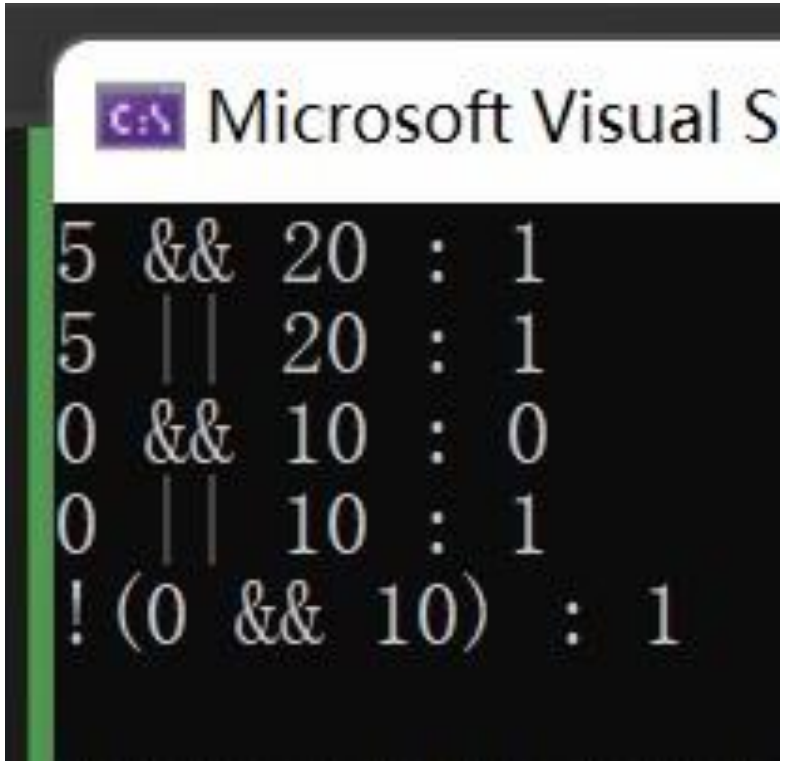
Logical Operators

Example

```
#include <stdio.h>
main()
{
        int a = 5;
        int b = 20;
        int c;

        c = a && b;
        printf("5 && 20 : %d\n",c);
        c = a || b;
        printf("5 || 20 : %d\n",c);

        a = 0;
        b = 10;
        c = a && b;
        printf("0 && 10 : %d\n", c);
        c = a || b;
        printf("0 || 10 : %d\n", c);
        c = !(a && b);
        printf("!(0 && 10) : %d\n", c);
}
```



0 for false
others for true

Assignment operators

lvalue = rvlue

```
int A;
A = 5;
```

modifiable lvalue

rvalue

```
int A;
5 = A;
```

```
a+b = 5;
```

```
const int A;
A = 5;
```

常变量

Assignment operators

lvalue = rvlue

lvaue can be variables
cannot be expressions, constant variables.

```
a+b = 5;
```

rvaue can be variables,
expressions, constant variables.

Assignment operators

=

```
int A, B, C;
A = B = C = 5;
```

- From right to left
- May not be allowed in other program languages

Assignment operators

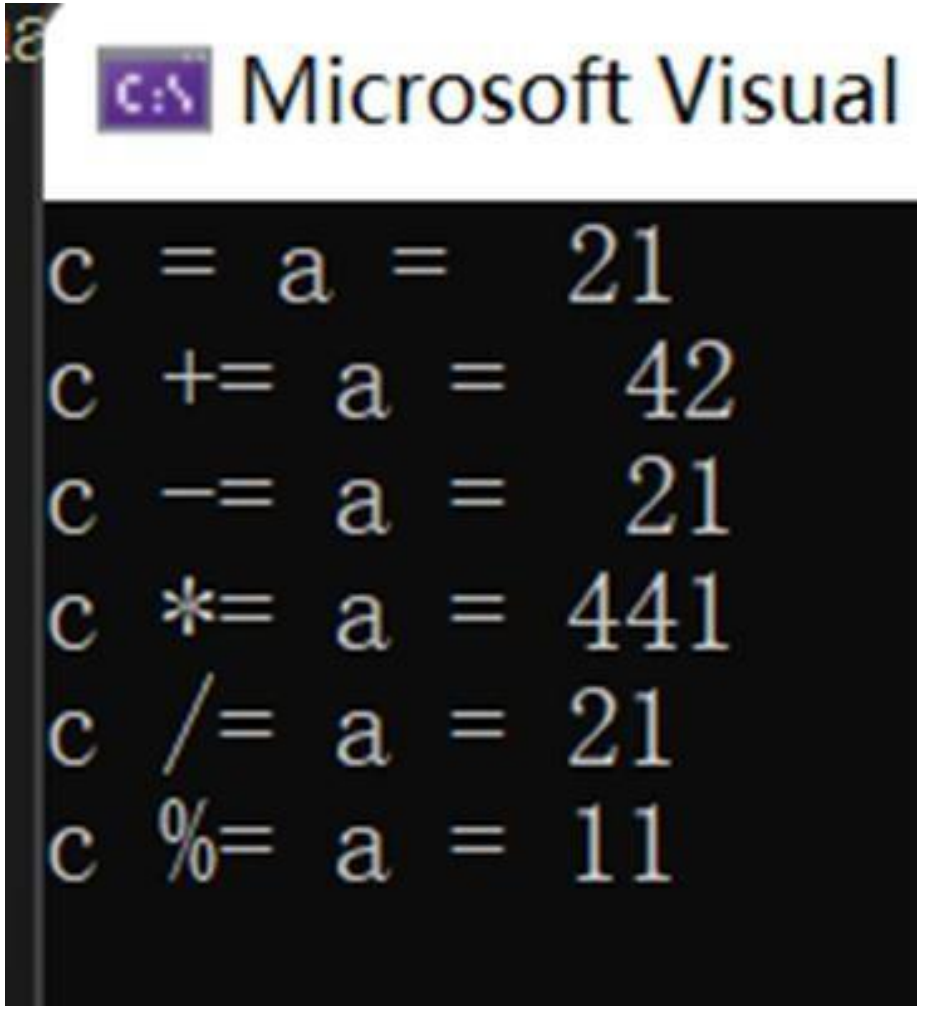
Define two variables: `int A = 5, B = 3;`

Operators	Description	Example
=	Simple assignment	B = B + A = 8
+=	Add and assign	B += A is B = B + A = 8
-=	Subtract and assign	B -= A is B = B – A = -2
*=	Multiply and assign	B *= A is B = B * A = 15
/=	Divide and assign	B /= A is B = B / A = 0
%=	Modulus and assign	B %= A is B = B % A = 3

Assignment operators

Example : assignment of an integer

```
#include <stdio.h>
main()
{
        int a = 21;
        int c;
        c = a;
        printf("c = a = %d\n", c);
        c += a;
        printf("c += a = %d\n", c);
        c -= a;
        printf("c -= a = %d\n", c);
        c *= a;
        printf("c *= a = %d\n", c);
        c /= a;
        printf("c /= a = %d\n", c);
        c = 200;
        c %= a;
        printf("c %%= a = %d\n", c);
}
```



Assignment operators

More example:

```
int a=12,b;
b=a+=a-=a*a;
```

Assignment operators

More example:

```
int a=12,b;
b=a+=a-=a*a;
```

- S1: a-=a*a;

```
a = a-a*a;
a = 12-144
a = -132
```

Assignment operators

More example:

```
int a=12,b;
b=a+=a-=a*a;
```

- S1: a-=a*a;

```
a = a-a*a;
a = 12-144
a = -132
```

- S2: a+=-132;

```
a =a +(-132)
b = -264
```

Miscellaneous operators

Define a variable: int A = 10; double B = -1.5;

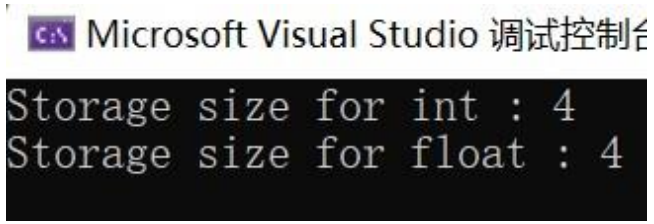
Operator	Description	Example
sizeof()	Return the size of variable (number of bytes)	sizeof(A) = 4 sizeof(B) = 8
&	Return the address of variable	&A = -2072708912 &B = -1602356112
?	Conditional expression	int flag = A>0 ? 1:0;
*	Pointer points to a variable	*A, *B

Few other important operators supported by C Language.

Miscellaneous operators

Example 1: use of sizeof()

```
#include <stdio.h>
main()
{
    int a = 10;
    float b = 3.14;
    printf("Storage size for int : %d \n", sizeof(a));
    printf("Storage size for float : %d \n", sizeof(b));
}
```



Miscellaneous operators

Example 2 : use of ?

```
#include <stdio.h>
main()
{
        int a = 10, b = 20;
        int c;
        c = a > b ? 1 : 0;
        printf("10 > 20 ? :%d\n", c);
        c = a < b ? 1 : 0;
        printf("10 < 20 ? :%d\n", c);
}
```

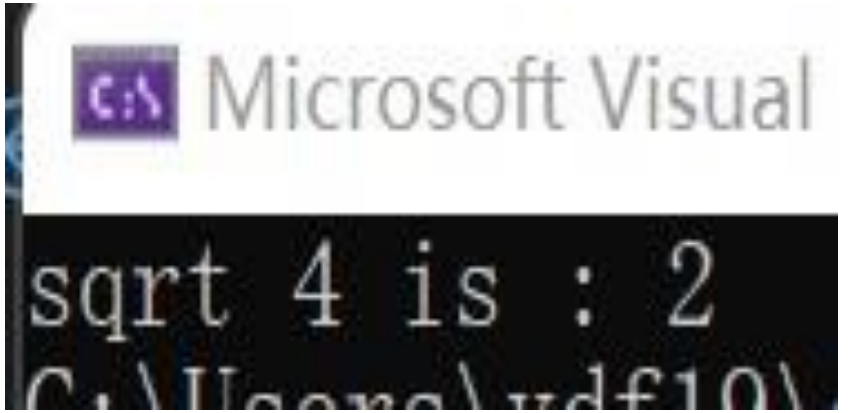
Miscellaneous operators

Example 3 : sqrt()

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

main()
{
        int a = 4;

        int b = sqrt(a);
        printf("sqrt 4 is : %d", b);
}
```

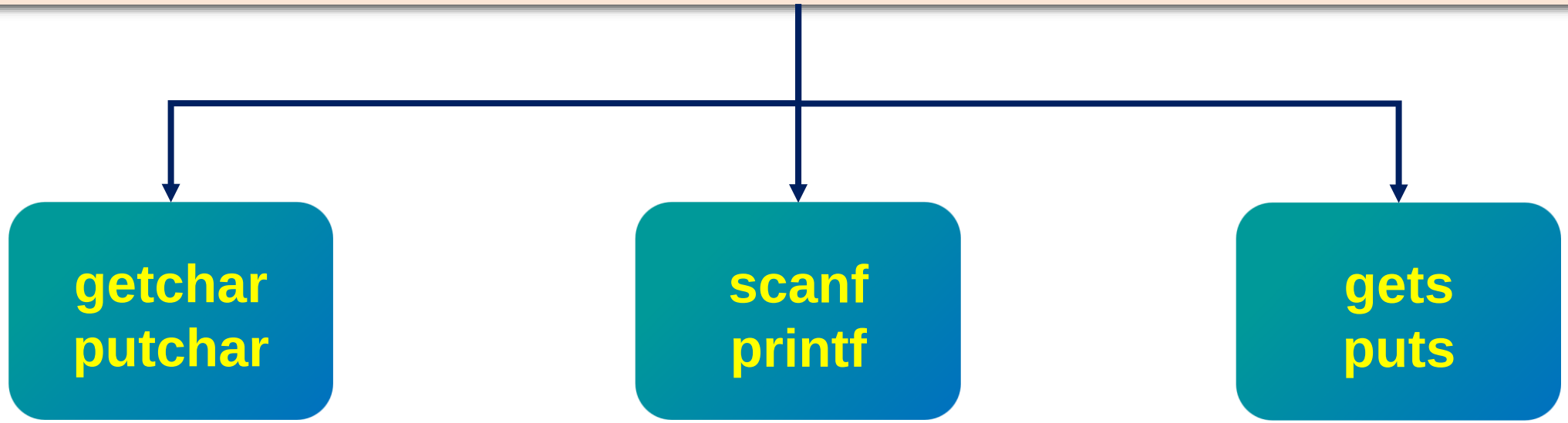


Content

- Data types and variables
- Operations and expressions
- Formatted Input/Output

IO

I/O defines how machine reads human's input and put on screen.



getchar() and putchar()

getchar() reads the next available single character and returns an integer representing the character in ASCII table.
putchar() puts the passed character on the screen

```
int c = getchar();
putchar(c);
```

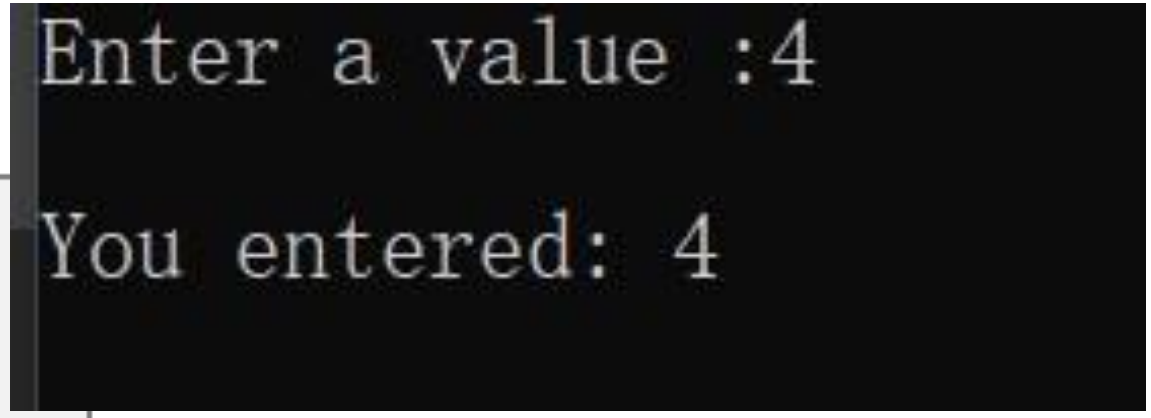
It reads and puts a single character!!!

getchar() and putchar()

Example 1: input an integer

```
#include <stdio.h>

int main( )
{
   int c;
   printf( "Enter a value :");
   c = getchar( );
   printf( "\nYou entered: ");
   putchar( c );
   return 0;
}
```



Here 4 is a character, with ID = 52 in ASCII!!!

getchar() and putchar()

Example 2: input a character

```c
#include <stdio.h>

int main()
{
    char character;

    printf("Enter a character:");
    character = getchar();
    printf("character = ");
    putchar(character);

    return (0);
}
```

```
Enter a character:d
character = d
```

getchar() and putchar()

Example 3: input two characters

```
#include "stdio.h"
int main()
{
    char c,d;
    printf("please input two
characters:\n");
    c=getchar();
    putchar(c);
    putchar('\n');
    d=getchar();
    putchar(d);
    putchar('\n');
    printf("character1 = %c\n",c);
    printf("character2 = %c\n",d);
    return(0);
}
```

```
please input two characters:
sd
s
d
character1 = s
character2 = d
```

gets() and puts()

gets() reads a string (a group of characters) from user and puts it into a buffer.
puts() shows the string on the screen

```
gets(char *s);
puts(char *s);
```

It reads and puts a group of characters!!!

gets() and puts()

Example : input a group of characters

```
#include <stdio.h>
int main( )
{
  char str[20]; // length of array is 20
  printf( "What's your name?\n");
  gets( str );
  printf( "\nYour name: ");
  puts( str );
  return 0;
}
```

```
What's your name?
Alex

Your name: Alex
```

scanf() and printf()

- Historically, they were not part of the definition of C

```
#include<stdio.h>
```

- Different versions of `scanf()` and `printf()` for different C versions

scanf() and printf()

scanf() reads the user input stream and scans it according to the provided format.
printf() writes to the output scream according to the format

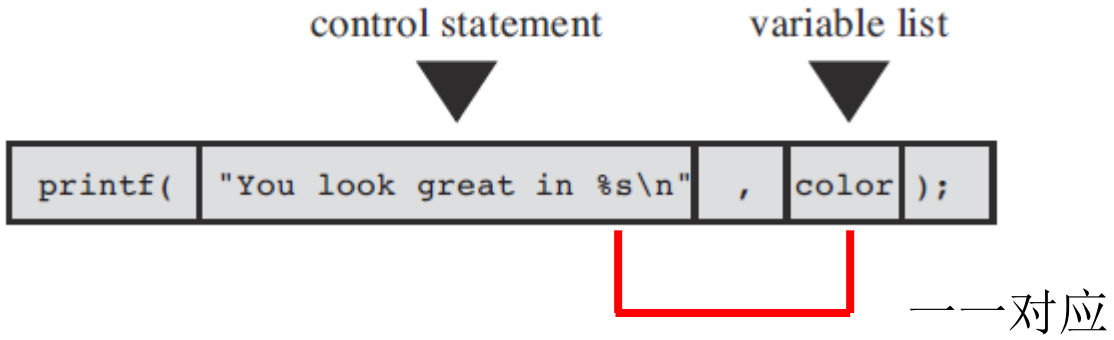
```
scanf([formatted text], [arguments]);
printf([formatted text], [arguments]);
```

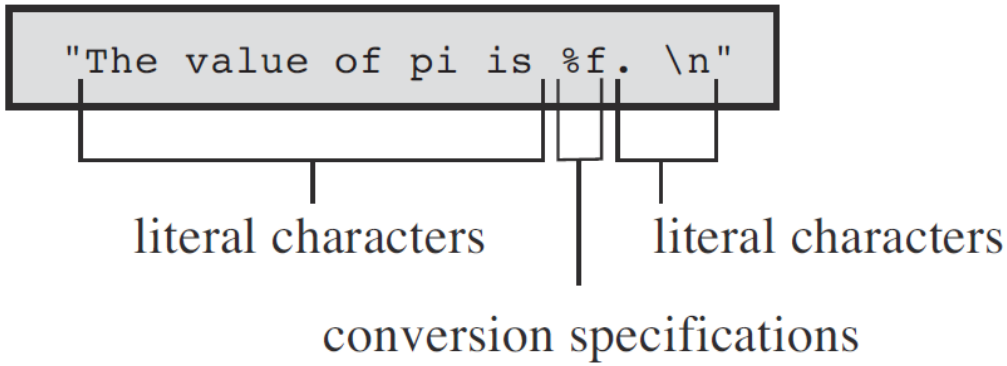
Formatted by specifiers
- %d int
- %f float
- %c char

f means formatted!!!

printf()

```
printf([formatted text], [arguments]);
```





（转换说明）

```
printf("Hello world!\m");
printf("%c%d\n", '$', 2*cost);
```

```
printf("The score was Squids %d, Slugs %d.\n",
        score1);
```

printf()

Conversion	Output Specification
%a	Floating-point number, hexadecimal digits and p-notation (C99/C11).
%A	Floating-point number, hexadecimal digits and P-notation (C99/C11).
%c	Single character.
%d	Signed decimal integer.
%e	Floating-point number, e-notation.
%E	Floating-point number, e-notation.
%f	Floating-point number, decimal notation.
%g	Use %f or %e, depending on the value. The %e style is used if the exponent is less than –4 or greater than or equal to the precision.
%G	Use %f or %E, depending on the value. The %E style is used if the exponent is less than –4 or greater than or equal to the precision.
%i	Signed decimal integer (same as %d).
%o	Unsigned octal integer.
%p	A pointer.
%s	Character string.
%u	Unsigned decimal integer.
%x	Unsigned hexadecimal integer, using hex digits 0f.
%X	Unsigned hexadecimal integer, using hex digits 0F.
%%	Prints a percent sign.

printf()

%d:以带符号的十进制形式输出整数
%o:以八进制无符号形式输出整数
%x:以十六进制无符号形式输出整数
%u:以无符号十进制形式输出整数
%c:以字符形式输出，只输出一个字符
%s:输出字符串
%f:以小数形式输出单，双精度数，隐含输出六位小数
%e:以指数形式输出实数
%g:选用%f或%e格式中输出宽度较短的一种格式，不输出无意义的0

Conversion specification modifiers for printf()

Modifier	Meaning
flag	The five flags (–, +, space, #, and 0) are described in Table 4.5. Zero or more flags may be present.
	Example: `"%-10d"`.
digit(s)	The minimum field width. A wider field will be used if the printed number or string won't fit in the field.
	Example: `"%4d"`.
.digit(s)	Precision. For `%e`, `%E`, and `%f` conversions, the number of digits to be printed to the right of the decimal. For `%g` and `%G` conversions, the maximum number of significant digits. For `%s` conversions, the maximum number of characters to be printed. For integer conversions, the minimum number of digits to appear; leading zeros are used if necessary to meet this minimum. Using only `.` implies a following zero, so `%.f` is the same as `%.0f`.
	Example: `"%5.2f"` prints a `float` in a field five characters wide with two digits after the decimal point.
`h`	Used with an integer conversion specifier to indicate a `short int` or `unsigned short int` value.
	Examples: `"%hu"`, `"%hx"`, and `"%6.4hd"`.
`hh`	Used with an integer conversion specifier to indicate a `signed char` or `unsigned char` value.
	Examples: `"%hhu"`, `"%hhx"`, and `"%6.4hhd"`.

Conversion specification modifiers for printf()

Modifier	Meaning
	Examples: `"%hhu"`, `"%hhx"`, and `"%6.4hhd"`.
`j`	Used with an integer conversion specifier to indicate an `intmax_t` or `uintmax_t` value; these are types defined in `stdint.h`.
	Examples: `"%jd"` and `"%8jX"`.
`l`	Used with an integer conversion specifier to indicate a `long int` or `unsigned long int`.
	Examples: `"%ld"` and `"%8lu"`.
`ll`	Used with an integer conversion specifier to indicate a `long long int` or `unsigned long long int`. (C99).
	Examples: `"%lld"` and `"%8llu"`.
`L`	Used with a floating-point conversion specifier to indicate a `long double` value.
	Examples: `"%Lf"` and `"%10.4Le"`.

提醒：

新手尽量使用简单格式

scanf()-conversion specifier

Table 4.6 **ANSI C Conversion Specifiers for `scanf()`**

Conversion Specifier	Meaning
`%c`	Interpret input as a character.
`%d`	Interpret input as a signed decimal integer.
Conversion Specifier	**Meaning**
`%e, %f, %g, %a`	Interpret input as a floating-point number (`%a` is C99).
`%E, %F, %G, %A`	Interpret input as a floating-point number (`%A` is C99).
`%i`	Interpret input as a signed decimal integer.
`%o`	Interpret input as a signed octal integer.
`%p`	Interpret input as a pointer (an address).
`%s`	Interpret input as a string. Input begins with the first non-whitespace character and includes everything up to the next whitespace character.
`%u`	Interpret input as an unsigned decimal integer.
`%x, %X`	Interpret input as a signed hexadecimal integer.

printf() and scanf()*

```
int main(void)
{
  unsigned width, precision;
  int number = 256;
  double weight = 242.5;
  printf("Enter a field width:\n");
  scanf("%d", &width);
  printf("The number is :%*d:\n", width, number);
  printf("Now enter a width and a precision:\n");
  scanf("%d %d", &width, &precision);
  printf("Weight = %*.*f\n", width, precision, weight);
  printf("Done!\n");
  return 0;
}
```

Enter a field width:
6
The number is : 256:
Now enter a width and a precision:
8 3
Weight = 242.500
Done!

printf() and scanf()*

```
/* skiptwo.c -- skips over first two integers of input
*/
#include <stdio.h>
int main(void)
{
  int n;
  printf("Please enter three integers:\n");
  scanf("%*d %*d %d", &n);
  printf("The last integer was %d\n", n);
  return 0;
}
```

Please enter three integers:
2013 2014 2015
The last integer was 2015

This skipping facility is useful if, for example, a program needs to read a particular column of a file that has data arranged in uniform columns.

scanf() and printf()

Example 1: input 2 integers and make calculation

```
#include<stdio.h>
int main(void)
{ int num1;
  int num2;
  int num3=0;
  printf("please enter number1:");
  scanf("%d",&num1);
  printf("please enter number2:");
  scanf("%d",&num2);
  num3=num1+num2;
  printf("number1 + number2 = %d\n",num3);
  return 0;
}
```

```
please enter number1:4
please enter number2:5
number1 + number2 = 9
```

int num1;

&num1

Get address of num1

scanf() and printf()

Example 2: input 2 integers in char and `int` formats

```
#include <stdio.h>
int main(void)
{
  char str[100];
  int i;

  printf( "Enter two value :");
  scanf("%s %d", str, &i);
  printf( "\nYou entered: %s, %d ", str, i);
  return 0;
}
```

```
Enter two value :67 76

You entered: 67, 76
```

int i; **char str[100];**

&i str

Get address of i

scanf() and printf()

Example 3: input different types of data

```
3 h 0.9
a = 3,  b = 0.90, ch = h
```

```
#include<stdio.h>
int main(void)
{   int a;
    char ch;
    float b;
    scanf("%d %c %f",&a,&ch,&b);
    printf("a = %d, b = %.2f,ch = %c\n", a, b, ch);
return 0;
}
```

Content

- **Bit and byte**
- **Data types and variables**
- **Operations and expressions**
- **Formatted Input/Output**

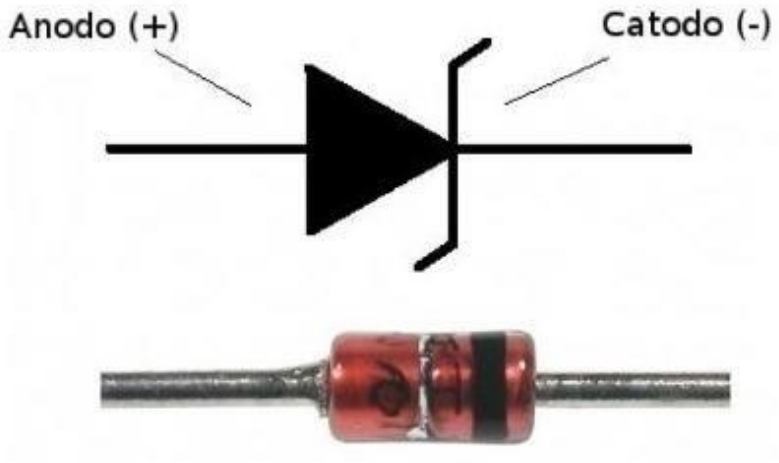
Bit and byte

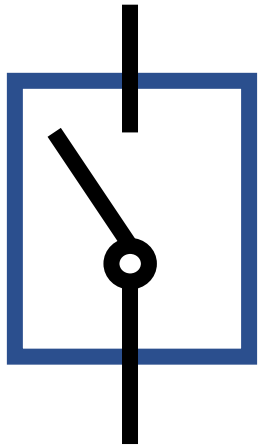
Bit (位)
The smallest unit for storage (atomic), **0 or 1**

Byte (字节)
The smallest unit for information storage, **1 byte = 8 bits**

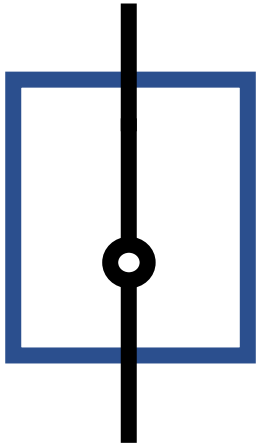
Bit

Computer is nothing but a vast collection of **diodes (on and off)**, denoting the state of 0 and 1.





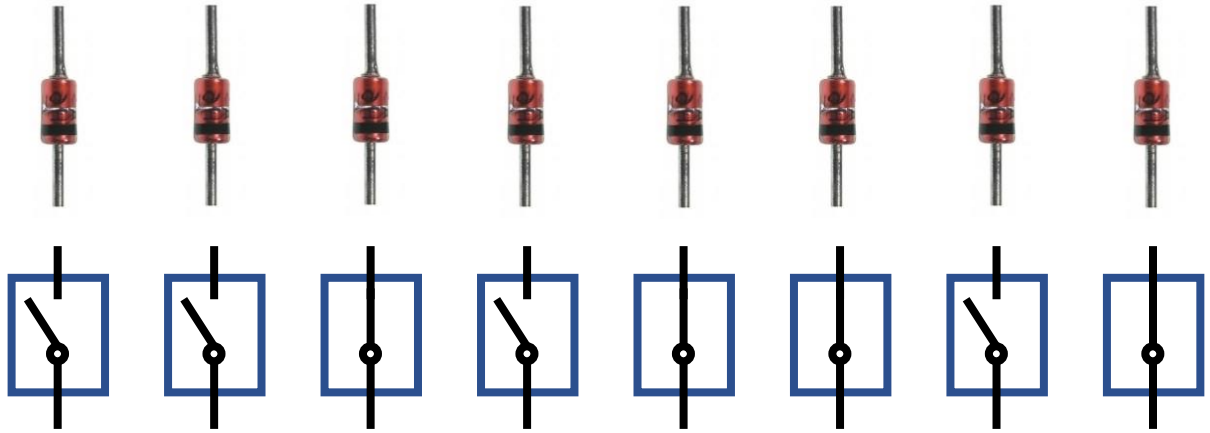
Off
“0”
False/No



On
“1”
True/Yes

Byte

1 byte = 8 bits



More diodes = More bits

More complex information

- 1024 byte = 1 KB (Megabyte)
- 1024 KB = 1 MB (Megabyte)
- 1024 MB = 1 GB (Megabyte)
- 1024 GB = 1 TB (Megabyte)
- 1024 TB = 1 PB (Megabyte)