Iteration

The identifiers 标识符 biāozhìfú

- We choose our own identifiers to name memory cells that will hold data and program results and to name operations that we define
- The syntax rules and some valid identifiers follows:
    i. An identifier must consist only of letters, digits, and underscores(`_`).
    ii. An identifier cannot begin with a digit.
    iii. A C reserved word cannot be used as an identifier.
    iv. An identifier defined in a C standard library should not be redefined.

Expression表达式

- In programming, an expression is any legal combination of symbols that represents a value

```
amount = x * (1 + 0.033) * (1 + 0.033) * (1 + 0.033);
total = 57;
count = count + 1;
value = (min / 2) * lastValue;
```

Expression表达式

- In programming, an expression is any legal combination of symbols that represents a value

```
amount = x * (1 + 0.033) * (1 + 0.033) * (1 + 0.033);
total = 57;
count = count + 1;
value = (min / 2) * lastValue;
```

- 对表达式的理解：
  i. 表达了计算的过程；
  ii. 表达了计算后的值（从而可以作为整体参加其他计算）
    - 表达式是由表达式组成的

Operator运算符（算子）

- A symbol that represents a specific action. For example, a plus sign (`+`) is an operator that represents addition. The basic mathematic operators are `+` addition, `-` subtraction, `*` multiplication, `/` division
- Assignment is an operator in C
- Operands(运算数) are the objects that are manipulated in an expression

Operator

```
int sides = 4;
sides = 7;
sides = sides + 5;
```

try

```
result = 12 + 6 / 2;
result = (12 + 6) / 2;
result = 4 * ((12 - 4) / 2);
```

Precedence(优先级)

Precedence	Operator	Meaning	Associative	Example
1	`+`	unary unchanged	`<-`	`a*+b`
1	`-`	unary minus	`<-`	`a*-b`
2	`*`	multiple	`->`	`a*b`
2	`/`	divide	`->`	`a/b`
2	`%`	remainder	`->`	`a%b`
3	`+`	plus	`->`	`a+b`
3	`-`	minus	`->`	`a-b`
4	`=`	assignment	`<-`	`a=b`

Unary(单目) operators

- Operators with only one operant: `+`, `-`

```
int a = 10;
int b = -20;
printf("%d", a * - b);
```

Operator assignment `=`

- Assignment is a sort of operating, thus has a result and a side effect
- The result of `a=6` is the value `a` gets, i.e. `6`
- `a=b=6` —> `a=(b=6)`
- The side effect is the fact that `b` gets `6` and `a` gets what `b` is

Result vs side-effect

- An operator may have a result and a side-effect
- All operators have a result
- Assignment operators have both result and side-effect

“Embedded assignment”

```
int a = 6;
int b;
int c = 1+(b=a);
```

- ❌ not read-friendly
- easy to make error

Association 结合关系

- Mostly: from left to right
- Unary and assignment: from right to left

```
result = a = b = 3 + c;
result = 2;
result = (result = result * 2) * 6 * (result = 3 + result);
```

- 这样的表达式太复杂，不容易阅读和理解，容易造成读程序时的误解。所以，要避免写出这样的复杂表达式来的。这个表达式应该被拆成若干个表达式，然后以明显的正确的顺序来进行计算。

More on `if`

A two alternative `if` statements

- The `if` statement below has two alternatives:

```
if (crsr_or_frgt == 1) {
    printf("Cruiser\n");
} else {
    printf("Frigate\n");
}
```

It prints either `Cruiser` or `Frigate`, depending on the value stored in the type `int` variable `crsr_or_frgt`

- The `if` statement that follows has one alternative; it prints the message `Cruiser` only when `crsr_or_frgt` has the value 1. Regardless of whether `Cruiser` is printed or not, the message `Frigate` is to be printed.

```
if (crsr_or_frgt == 1) {
    printf("Cruiser\n");
}
printf("Frigate\n");
```

`if` Statement (two alternatives)

FORM:

```
if (condition)
    statementT;
else
    statementF;
```

`if` Statement (two alternatives)

EXAMPLE:

```
if (x >= 0.0)
    printf("positive\n");
else
    printf("negative\n");
```

- INTERPRETATION: If condition evaluates to true (a nonzero value), then `statementT` is executed and statementF is skipped; otherwise, `statementT` is skipped and `statementF` is executed.

Integer Partition

Integer Partition

- Given a thee figures (3-digit number), print out the sum of each digit
- IPO
  - Input: a number `x`
  - Output: the `sum` of each digit
  - Process: decompose digits and sum them up

Basic Tech

- `%10` —> get the last digit
- `/10` —> get rid of the last digit
- Repeat the process until there is not digit left —> `==0`

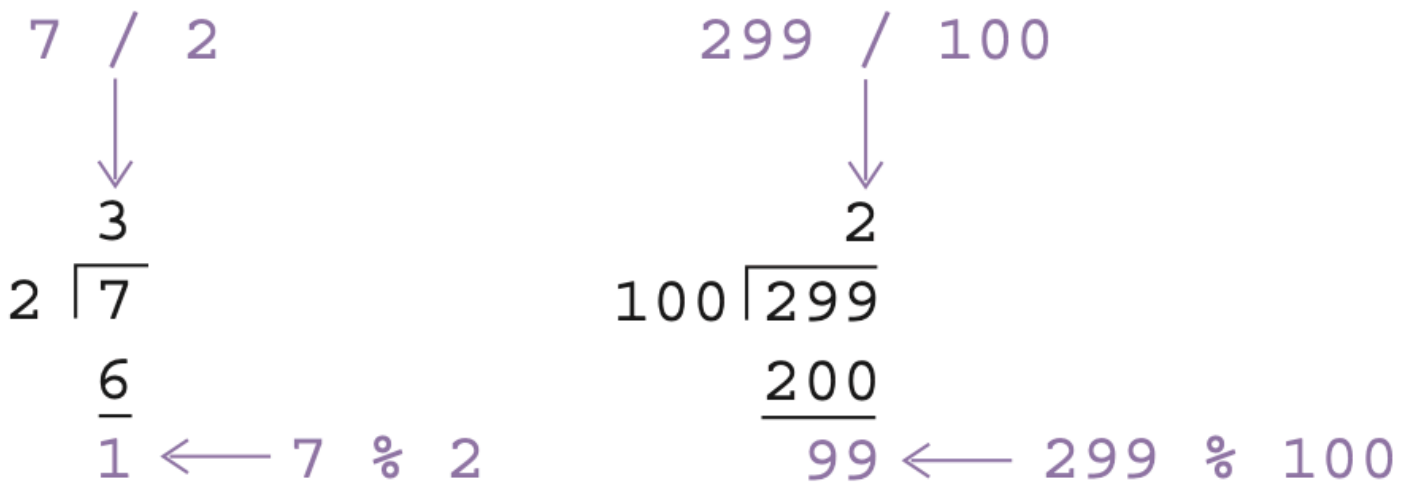
Operator `/`

- The result of the operator `/` is an integer when both operands are integers

```
15 / 2 --> 7
20 / 3 --> 6
```

Operators `%`

- The remainder operator (`%`) returns the integer remainder of the result of dividing its first operand by its second.



- $m = (m/n) \cdot n + (m\%n)$

Count the number of digits

- Given a non-negative integer, print out the number of digits.
- IPO
  - Input: a non-negative integer `x`
  - Output: the number of digits `n`
  - Process?

Process

1. x = x/10
2. n = n +1
3. If x==0 break
4. Repeat 1-4

Manually simulation

- List all the variables in a table, and write down the values during the execution of the program

Validation of your code

- Usually, boundary data, like number at the edge of valid range, special multiples, are used to test code.
  - single digit
  - 10
  - 0
  - negative numbers

Debug

- Put `printf` as needed to print out the values

```c
int x;
int n = 0;

scanf("%d", &x);

while (1) {
    printf("x=%d, n=%d\n", x, n);
    n++;
    x = x/10;
    if ( x==0 ) {
        break;
    }
    printf("%d\n", n);
}
```

Lab 1

5-1: 分解整数

Real Numbers

Height 5-7?

- American use feet and inches. If someone told you he is 5-7, what is that in meters?

$$(5 + \frac{7}{12}) \times 0.3048 = 1.7018m$$

A wrong program

```c
#include <stdio.h>

int main()
{
    int foot;
    int inch;
    scanf("%d %d", &foot, &inch);
    printf("%d\n", (foot + inch/12)*0.3048);
}
```

- The result of `(foot + inch/12)*0.3048` is a number with decimal part, which is not an integer.
- The `%d` in `printf`'s format string indicates the value to be an integer. However it is not an integer in that case.
- To specify a number with decimal part, use `%f` instead.

V1.0

```c
#include <stdio.h>

int main()
{
    int foot;
    int inch;
    scanf("%d %d", &foot, &inch);
    printf(“%f\n", (foot + inch/12)*0.3048);
}
```

Try

Why always 1.524?

Because...

- When applied to two positive integers, the division operator (`/`) computes the integral part of the result of dividing its first operand by its second
- Result of two integers is always an integer

  `10/3*3=>?`
- 10 and 10.0 are totally different in C
- 10.0 is a floating-point number (real number)

Floating-point number

- Aka real number. The term “floating-point” means the decimal point is floating, not at a fixed position. It is a way to represent real number in computer, and is the term for real number in the C.

Modification

From

```
(foot + inch / 12) * 0.3048;
```

to

```
(foot + inch / 12.0) * 0.3048;
```

- When an integer meets a floating with one operator, the integer will be turned into a corresponded floating. Thus the calculation takes place between two floating and the result is going to be a floating.

V2.0

```c
#include <stdio.h>
#define INCH_TO_CM_FACTOR 0.3048

int main()
{
    int foot;
    int inch;
    scanf("%d %d", &foot, &inch);
    printf("%f\n", (foot + inch/12.0)*INCH_TO_CM_FACTOR);
}
```

- the `INCH_TO_CM_FACTOR` here is a macro that represents a fixed value
- It is a good practice to avoid "magic number" in the code

`double`

- `inch` is an `int` variable. If we modify the keyword `int` into `double`, it is going to be a double floating
- `double` means double precision. In this way, C has `double` and `float`

V3.0

```c
#include <stdio.h>
#define INCH_TO_CM_FACTOR 0.3048

int main()
{
    double foot;
    double inch;
    scanf("%lf %lf", &foot, &inch);
    printf("%f\n", (foot + inch/12.0)*INCH_TO_CM_FACTOR);
}
```

Data types

- A data type is a set of values and a set of operations on those values. Knowledge values and operations that can be performed on those values of the data type of an item (a variable or value) enables the C compiler to correctly specify operations on that item.
- We use the standard data types `double` and `int` as abstractions for the real numbers and integers (in the mathematical sense).

- Integers

```
int
printf("%d",...);
scanf("%d",...);
```

- Real numbers

```
double
printf("%f",...);
scanf("%lf",...);
```

Mixed-type assigment

```
int m = 5,n = 6;
double y;
//...
y = m / n;
```

- What will be `y`?

Lab 2

5-2: 求平均

Library Functions

Predefined functions and code reuse

- A primary goal of software engineering is to write error-free code. Code reuse, reusing program fragments that have already been written and tested whenever possible, is one way to accomplish this goal. Stated more simply, “Why reinvent the wheel?”
- C promotes reuse by providing many predefined functions that can be used to perform mathematical computations.

- We can use the C functions `pow` (power) and `sqrt` (square root) to compute the roots of a quadratic equation in x of the form:

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

- the two roots are defined as:

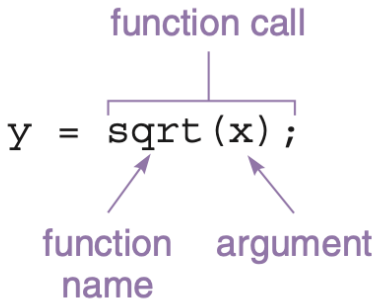
$$root_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$root_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

when the discriminant $(b^2 - 4ac)$ is greater than zero.

`sqrt()`

- C's standard math library defines a function named `sqrt` that performs the square root computation. The function call in the assignment statement:



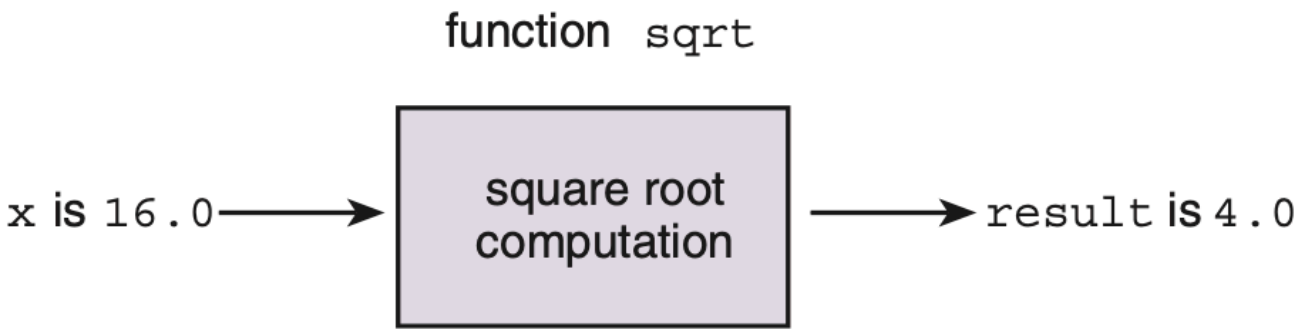
  activates the code for function `sqrt`, passing the argument `x` to the function. You activate a function by writing a function call. After the function executes, the function result is substituted for the function call.

- To use this function, another header file should be included at the beginning:

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

Function

- A function can be thought of as a “black box” that receives one or more input values and automatically returns a single output value.
- Figure below illustrates this for the call to function sqrt. The value of `x` (`16.0`) is the function input, and the function result, or output, is $\sqrt{16.0}$ (result is `4.0`).



Using functions

```
first_sqrt = sqrt(first);
second_sqrt = sqrt(second);
sum_sqrt = sqrt(first + second);
a_sqrt = sqrt(sqrt(first));
```

- substitution: the result of the function is to be used to replace the function call

$$root_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$
$$root_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

- we can use these assignment statements to assign values to `root_1` and `root_2`.

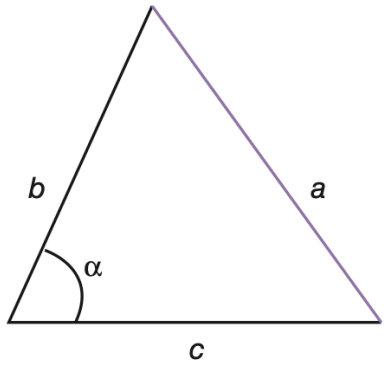
```
/* Compute two roots, root_1 and root_2, for disc > 0.0 */
disc = pow(b,2) - 4 * a * c;
root_1 = (-b + sqrt(disc)) / (2 * a);
root_2 = (-b - sqrt(disc)) / (2 * a);
```

- In the first assignment statement above, the expression begins with `pow(b, 2)`, which calls function pow with `b` and `2` as arguments; the function result (b^2) is substituted for the function call when the expression is evaluated.

Triangle

- If we know the lengths of two sides (b and c) of a triangle and the angle between them in degrees (α), we can compute the length of the third side (a) using the following formula:

$$a^2 = b^2 + c^2 - 2bc \cdot cos\ \alpha$$



deg vs rad

- To use the math library cosine function (`cos`), we must express its argument angle in radians instead of degrees.
- To convert an angle from degrees to radians, we multiply the angle by $\frac{\pi}{180}$. `PI` represents the constant π, as defined in <math.h>:
  - `cos(alpha * PI / 180.0))`

Could you write down the C code to calculate a from b, c and α?

User Defiend Functions

Square

- `pow()` is a `double` function using Taylor expansion
- power to 2 can be calculated much more easier as `x*x`
- We can define our own square function

```
double square(double x)
{
    return x*x;
}
```

Using `square()`

- Once defined, it can be used in our own program:

```c
#include <stdio.h>
#include <math.h>
double square(double x)
{
    return x*x;
}
int main()
{
    int x;
    scanf("%d", &x);
    double disc = square(b) - 4 * a * c;
    double root_1 = (-b + sqrt(disc)) / (2 * a);
    double root_2 = (-b - sqrt(disc)) / (2 * a);
    printf("%.2f %.2f\n"m root_1, root_2);
}
```

Circumference and area of a circle

```c
/*
Computes the circumference of a circle with radius r. Pre: r is defined and is > 0.
PI is a constant macro representing an approximation of pi.
*/
double find_circum(double r) {
    return 2.0 * PI * r;
}
/*
Computes the area of a circle with radius r. Pre: r is defined and is > 0.
PI is a constant macro representing an approximation of pi. Library math.h is included.
*/
double find_area(double r) {
    return PI * square(r);
}
```

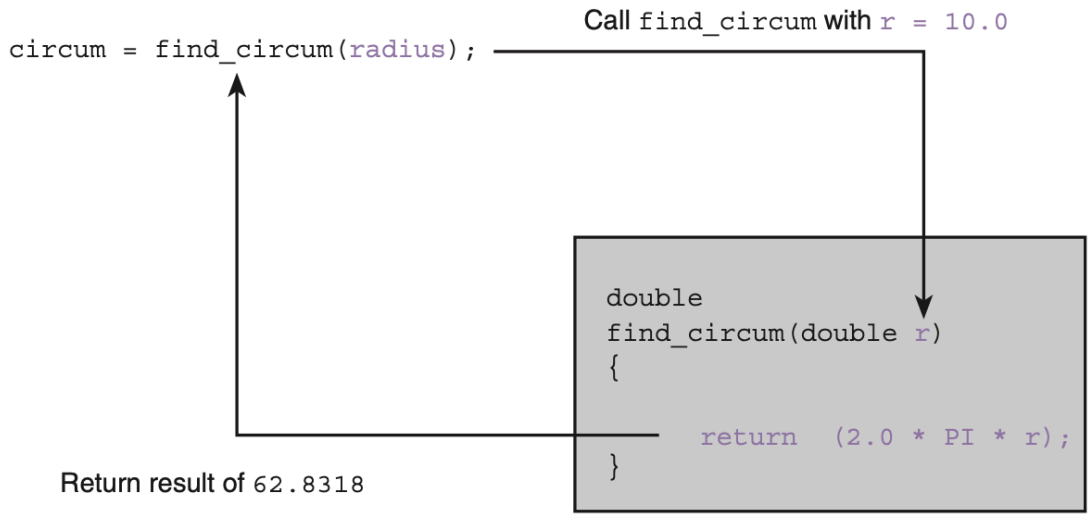
- Each function heading begins with a word like `double`, indicating that the function result is a real number
- Both function bodies consist of a single `return` statement. When either function executes, the expression in its `return` statement is evaluated and returned as the function's result. If `PI` is the constant macro 3.14159, calling function `find_circum` causes the expression `2.0 * 3.14159 * r` to be evaluated

Calling the function

- C substitutes the actual argument used in the function call for the formal parameter `r`. For the function call below:

```
radius = 10.0;
circum = find_circum(radius);
```

the actual argument, `radius`, has a value of 10.0, so the function result is $2.0 * 3.14159 * 10.0 = 62.8318$. The function result is then assigned to `circum`



Lab 3

6-1: Find circumference

What we've learned today

- Identifiers
- Precedence of the operators
- Two alternative if statement
- Partition of an integer
- `doube` : the real numbers
- Using library functions
- Defining our own functions