C/C++ Programming Language

CS205 Spring

Feng Zheng

2019.03.14

南方科技大学

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Content

- Brief Review
- Pointer (Second Half)
- Managing Memory for Data
- Loops and Relational Expressions
- Summary

Brief Review

Compound Types

- Array Types
- Strings
  - C-style String
  - string-class string
- Structure
  - Structure: struct
  - Union: union
  - Enumeration: enum

Address Types

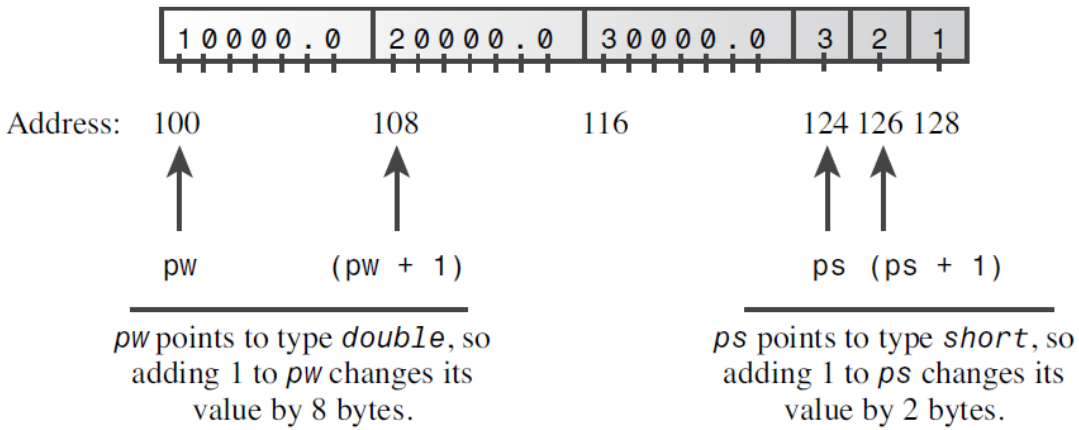
- Pointer (Half)
  - Address operator: &
  - Indirect value operator: *
  - Allocate memory: new
  - Release memory: delete

Pointer (Second Half)

Pointers, Arrays, and Pointer Arithmetic

- Adding one to a pointer variable increases its value by the number of bytes of the type to which it points
- Program example 10
  - ➢ You can use pointer names and array names in the same way
  - ➢ Differences between them
    - ① You can change the value of a pointer, whereas an array name is a constant
    - ② Applying the sizeof operator to an array name yields the size of the array, but applying sizeof to a pointer yields the size of the pointer

```
double wages[3] = {10000.0, 20000.0, 30000.0};
short stacks[3] = {3, 2, 1};
double * pw = wages;
short * ps = &stacks[0];
```



The Address of an Array

- Program example 11

  short tell[10];
  - tell is type pointer-to-short
  - &tell is type pointer-to-array of 10 shorts

  - short (*pas)[10] = &tell; // try to replace 10 by 20
  - (*pas) = tell is type pointer-to-short
  - pas=&tell is type pointer-to-array of 10 shorts

  - short* pas[10];
  - pas is an array of 10 pointers-to-short

- **&tell**

Summarizing Pointer Points

- Pointers
  - Declaring pointers
  - Assigning values to pointers
  - Dereferencing pointers: means referring to the pointed-to value
  - Distinguishing between a pointer and the pointed-to value
- Array names
  - Bracket array notation is equivalent to dereferencing a pointer
- Pointer arithmetic
- **Dynamic** binding and static binding for arrays

```
int size;
cin >> size;
int * pz = new int [size];    // dynamic binding, size set at run time
...
delete [] pz;                 // free memory when finished
```

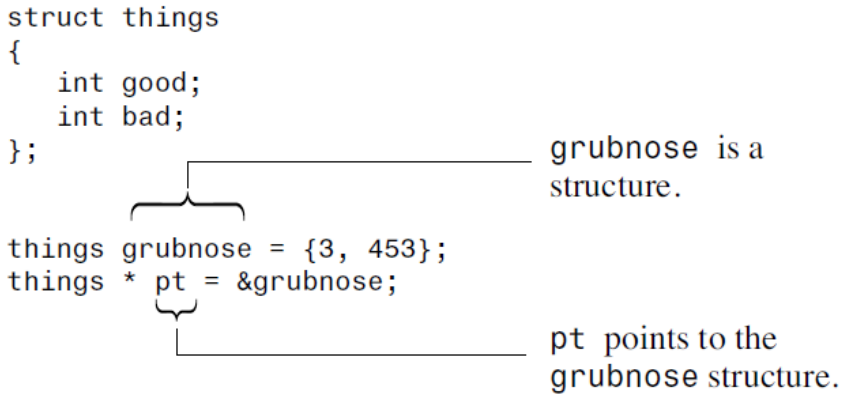
Using **new** to Create Dynamic Structures

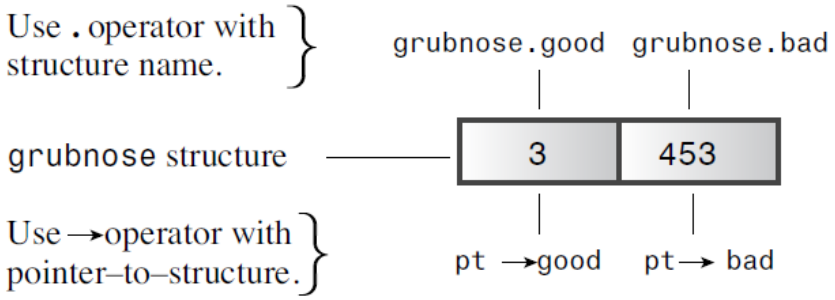
- Dynamic means the memory is allocated during runtime
  - Creating the structure
  - Accessing its members

```
inflatable * ps = new inflatable;
```

  - The arrow membership operator (->) of a hyphen and then a greater-than symbol

- Program example 12 (single)





An Example of Using **new** and **delete** for Functions

- Program example 13
  - Return the address of the string copy
  - It's usually not a good idea to put new and delete in separate functions

Managing memory for data

Automatic Storage

- Automatic Storage
  - Ordinary variables defined inside a function use automatic storage and are called automatic variables
  - They expire when the function terminates
  - Automatic variables typically are stored on a **stack**
  - A last-in, first-out, or LIFO, process

Static Storage

- Static Storage
  - Static storage is storage that exists throughout the execution of an entire program
  - Two ways

    ① Define it externally, outside a function

    ② Use the keyword static when declaring a variable

```
static double fee = 56.50;
```

Dynamic Storage

- Dynamic Storage
  - The new and delete operators provide a more flexible approach than automatic and static variables
  - Refer to as the free store or heap
  - Lifetime of the data is not tied arbitrarily to the life of the program or the life of a function

Combinations of Types

- Include arrays, structures, and pointers

- Program example 14: array of structures
  - const antarctica_years_end * arp[3] = {&s01, &s02, &s03};
  - const antarctica_years_end ** ppa = arp;
- Distinguish the following (again)
  - type_name * variable_name[10] ----- type_name (*variable_name)[10]

Array Alternatives

- The vector Template Class
  - Similar to the string class
  - It is a dynamic array
  - Use new and delete to manage memory
  - The vector identifier is part of the std namespace
- The array Template Class
  - The array identifier is part of the std namespace
  - The number of elements can't be a variable
- See Program Example 1
  - Comparing Arrays, Vector Objects, and Array Objects

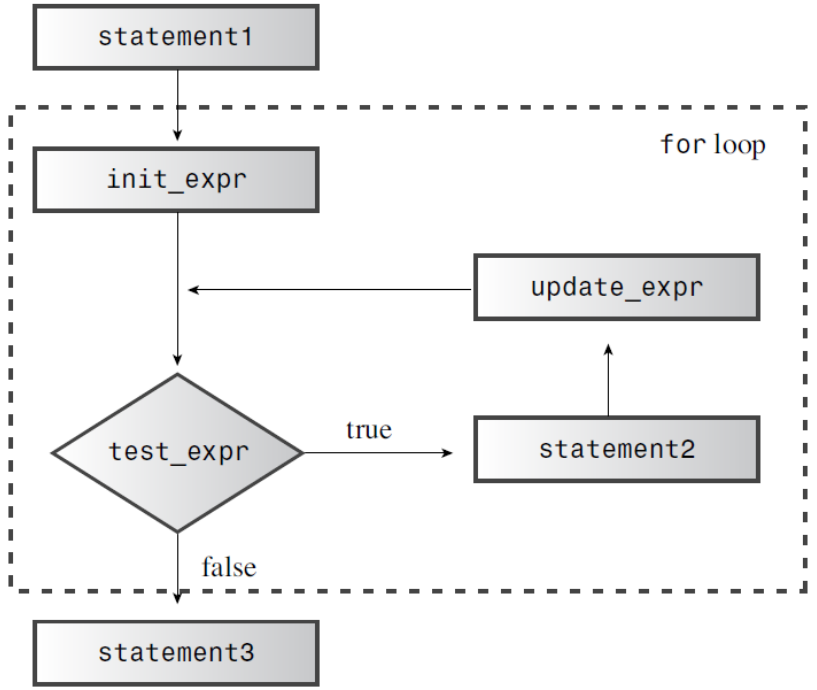
Loops and Relational Expressions

Introducing **for** Loops

- Why needs loop operations?
  - Perform repetitive tasks
- Parts of a for Loop
  - Setting a value initially
  - Testing whether the loop should continue
  - Executing the loop actions - body
  - Updating value(s) used for the test

```
for (initialization; test-expression; update-expression)
    body;
```

```
statement1
for (int_expr; test_expr; update_expr)
     statement2
statement3
```



Introducing **for** Loops

- Loops
  - The loop performs initialization just once
  - Test expression is a relational expression
  - Test expression is evaluated before each loop cycle
  - Update expression is evaluated at the end of the loop
- See program example 2
  - Increment operator: ++ operator (i = i + 1;)
- See program example 3
  - Decrement operator: -- operator (i = i - 1;)

More Examples

- See program example 4
  - Factorial definition
    - ✓ Zero factorial, written as 0!, is defined to be 1 (exclamation marks!)
    - ✓ The factorial of each integer being the product of that integer with the preceding factorial
- See program example 5
  - Changing the step size
- See program example 6
  - The increment (++) and decrement (--) operators

Expressions

- A C++ expression is a value or a combination of values and operators
- Every C++ expression has a value
  - A for control section uses three expressions
  - Relational expressions such as x < y evaluate to the bool values
  - Evaluating the expression is the primary effect
    - ✓ Evaluating x + 15 calculates a new value, but it doesn't change the value of x
    - ✓ But evaluating ++x + 15 does have a side effect because it involves incrementing x

Statements

- Statements
  - From expression to statement is a short step
  - You just add a semicolon
  - Declaration is not an expression
- Non-expressions and statements
  - Removing a semicolon from a statement does not necessarily convert it to an expression
    - ✓Return statements
    - ✓Declaration statements
    - ✓for statements

Side Effects and Sequence Points

- Side effect: occurs when evaluating an expression modifies something
- Sequence point: a point which all side effects are guaranteed to be evaluated before going on to the next step
- What's a full expression?
  - A test condition for a while loop
  - An expression portion of an expression statement
- The end of any full expression is a sequence point
  - Avoid statements of this kind

```
y = (4 + x++) + (6 + x++);
```

More for Increment/Decrement Operators

- Prefixing versus postfixing
  - Prefix form is more efficient
- The increment/decrement operators and pointers
  - Adding an increment operator to a pointer increases its value by the number of bytes in the type it points to
  - The prefix increment, prefix decrement, and dereferencing operators have the same precedence (from right to left)
  - Postfix increment and decrement operators have the same precedence, which is higher than the prefix precedence(from left to right)
- See program example 7

And More for Loops

- Combination assignment operators
  - Example: combined addition and assignment operator

Operator	Effect (L=left operand, R=right operand)
+=	Assigns L + R to L
-=	Assigns L - R to L
*=	Assigns L * R to L
/=	Assigns L / R to L
%=	Assigns L % R to L

- Compound statements, or blocks: {}
  - Program example 8
- More syntax tricks—the comma operator
- ```
  int i, j;  // comma is a separator here, not an operator
  ++j, --i   // two expressions count as one for syntax purposes
  ```
  ```

Relational Expressions

- C++ provides six relational operators to compare numbers
  - Exclamation mark

Operator	Meaning
<	Is less than
<=	Is less than or equal to
==	Is equal to
>	Is greater than
>=	Is greater than or equal to
!=	Is not equal to

Comparisons

- Program example 9
  - A mistake you'll probably make
  - = or ==
- Program example 10
  - Comparing C-style strings
  - strcmp(str1,str2)
- Program example 11
  - Comparing string class strings
  - Using relational symbol (!=)

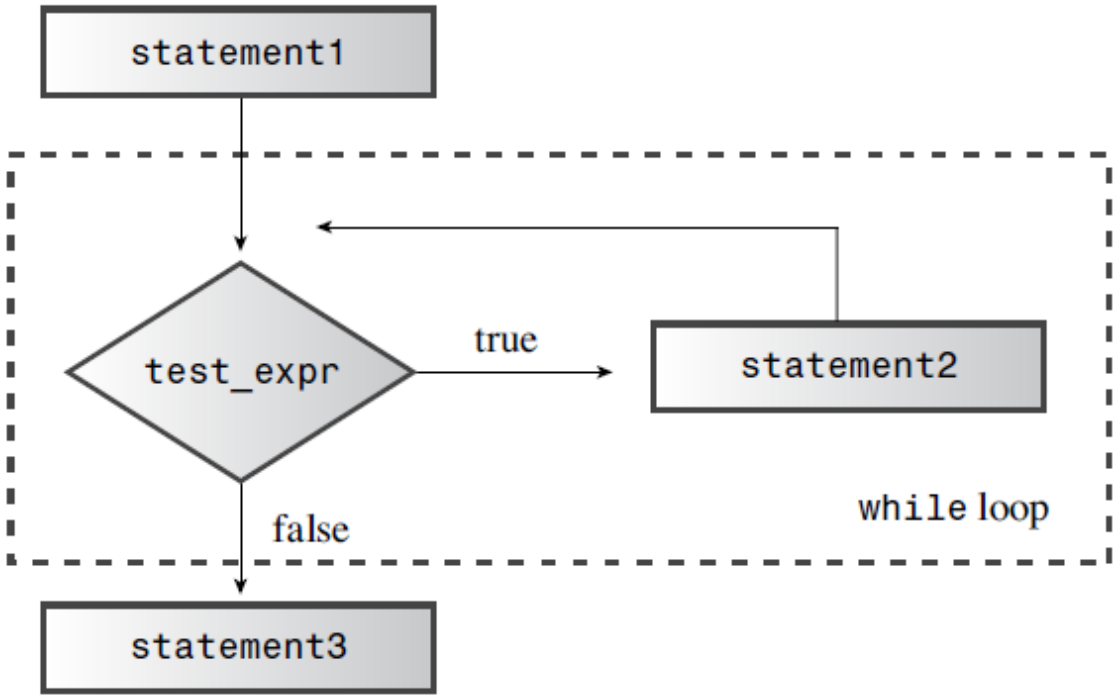
The while Loop

- while is entry-condition loop
- It has just a test condition and a body
  - Do something to affect the test-condition expression
- See Program example 12
  - Two types of condition expression

while (name[i] != '\0')

while (name[i])

```
statement1
while (test_expr)
      statement2
statement3
```



for Versus while

- In C++ the for and while loops are essentially equivalent

```
for (init-expression; test-expression; update-expression)
{
    statement(s)
}
```

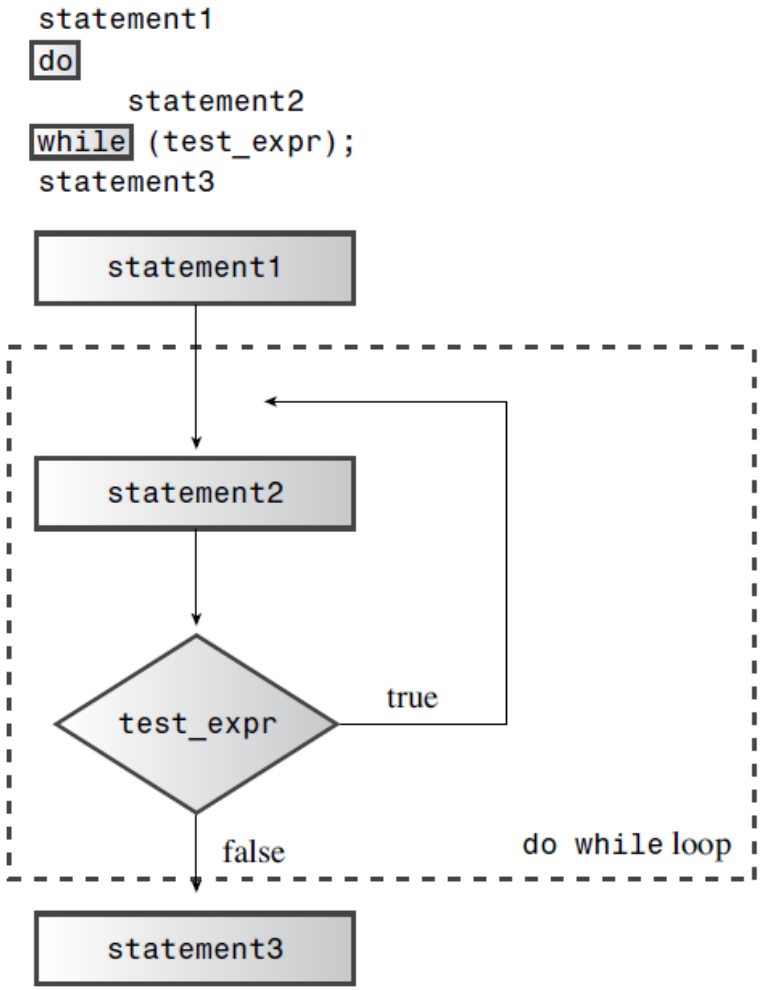
```
init-expression;
while (test-expression)
{
    statement(s)
    update-expression;
}
```

```
while (test-expression)
    body
```

```
for ( ;test-expression;)
    body
```

More Loops

- The do while Loop
  - It's an exit-condition loop
  - Such a loop always executes at least once
  - See Program example 13
- The range-based for loop (C++11)
  - See Program example 14
    - ✓ Colon symbol :
    - ✓ & symbol: reference variable
    - ✓ To modify the array contents



Loops and Text Input

- Using unadorned cin for input
  - When to stop?
    - ✓ A sentinel character
  - See program example 15
    - ✓ The program omit the spaces
    - ✓ Program and operating system both work
- cin.get(char) to the rescue
  - See program example 16
    - ✓ Read the space
    - ✓ Declare the argument as a reference

Nested Loops and Two-Dimensional Arrays

- Example:

int maxtemps[4][5];

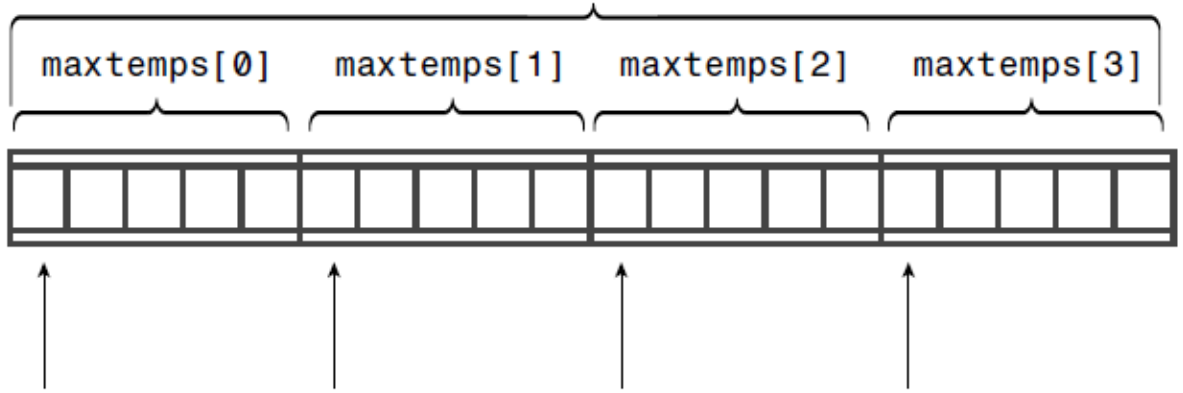
- See program example 17

maxtemps is an array of 4 elements

```
int maxtemps[4][5];
```

Each element is an array of 5 ints.

The maxtemps array

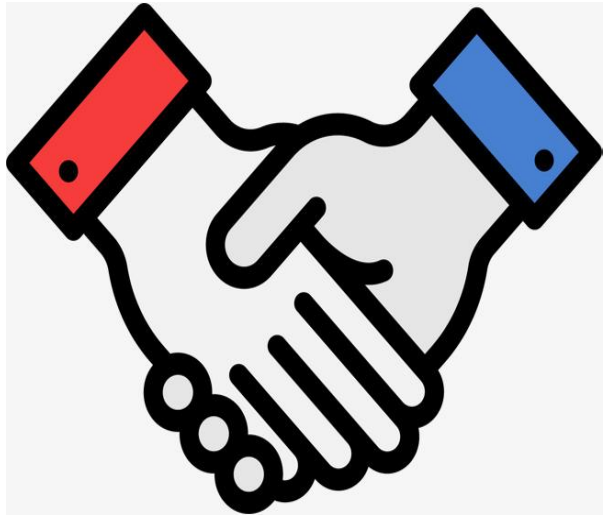


The maxtemps array viewed as a table:

	0	1	2	3	4
maxtemps[0]	0	maxtemps[0][0]	maxtemps[0][1]	maxtemps[0][2]	maxtemps[0][3]	maxtemps[0][4]
maxtemps[1]	1	maxtemps[1][0]	maxtemps[1][1]	maxtemps[1][2]	maxtemps[1][3]	maxtemps[1][4]
maxtemps[2]	2	maxtemps[2][0]	maxtemps[2][1]	maxtemps[2][2]	maxtemps[2][3]	maxtemps[2][4]
maxtemps[3]	3	maxtemps[3][0]	maxtemps[3][1]	maxtemps[3][2]	maxtemps[3][3]	maxtemps[3][4]

Summary

- Three varieties of loops: for, while, and do while
  - The loop test condition
  - Entry-condition loops
  - Exit-condition loops
- Relational expressions
  - Six relational operators
- Loops and text input
- A nested loop is a loop within a loop
  - Provide a natural way to process two-dimensional arrays



Thanks

zhengf@sustech.edu.cn