C/C++ Programming Language

CS205 Spring

Feng Zheng

Week 5

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Content

Brief Review

Content of Last Class

- Pointers
  - Address of array
  - new and delete operations

- Managing memory for data
  - Automatic memory
  - Dynamic memory
  - Static memory

Review of The Address of an Array

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

  - short (*pas)[10] = &tell;
  - (*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 (short *)

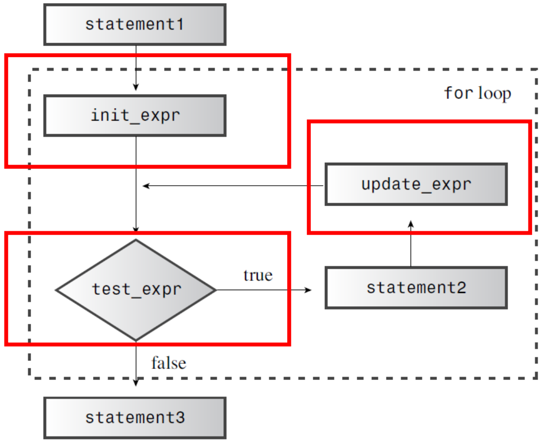
  - **&tell**

Loops and Relational Expressions

Introducing **for** Loops

- Why needs loop operations?
  - Perform repetitive tasks
  - Most tasks have the same process
- 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

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



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

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
- Run forloop.cpp
  - ➢ Increment operator: ++ operator (i = i + 1;)
- Run num_test.cpp
  - ➢ Decrement operator: -- operator (i = i - 1;)

More Examples

- Run formore.cpp
  - ➢ 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
- Run bigstep.cpp
  - ➢ Changing the step size
- Run plus_one.cpp
  - ➢ 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

primary effect & side effect

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

```
int fx = for (i = 0; i< 4; i++)
cout >> i; // not possible
```

Side Effects and Sequence Points

- Side effect: occurs when evaluating an expression (primary effect) **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

```
int x = 0;
//int y = (1 + x++) + (1 + x++);
//int y = 1 + x++ + 1 + x++;
```

More for Increment/Decrement Operators

- Prefixing versus postfixing: ++x, x++, --x, x--
  - 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)
- Run plus_one2.cpp

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: {}
  - ➢ Run blocks.cpp
- 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 in Test Expression

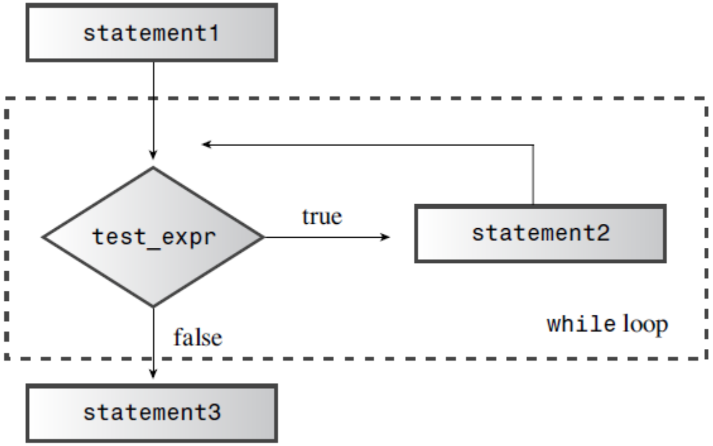
- Run equal.cpp
  - A mistake you'll probably make
  - = or ==
- Run compstr1.cpp
  - Comparing C-style strings
  - strcmp(str1,str2)
- Run compstr2.cpp
  - Comparing string class strings
  - Using relational symbol (!=)

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
- Run while.cpp
  - 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)
}
```

赋值语句永为true.

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

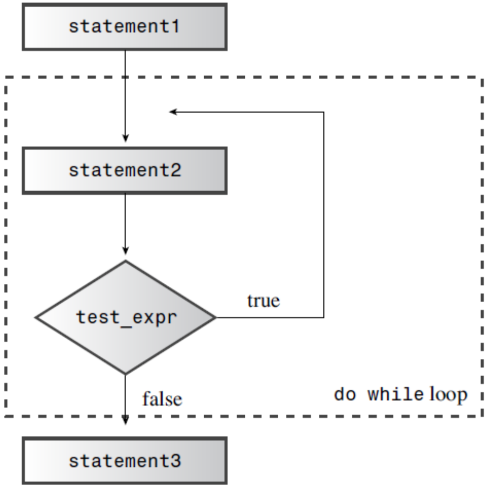
```
while (test-expression)
    body
```

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

More Loops

- The do while Loop
  - It's an exit-condition loop
  - Such a loop always executes at least once
  - Run dowhile.cpp
- The range-based for loop (C++11)
  - Run range_based.cpp
    - ✓ Colon symbol :
    - ✓ & symbol: reference variable
    - ✓ To modify the array contents

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



Example: Loops and Text Input

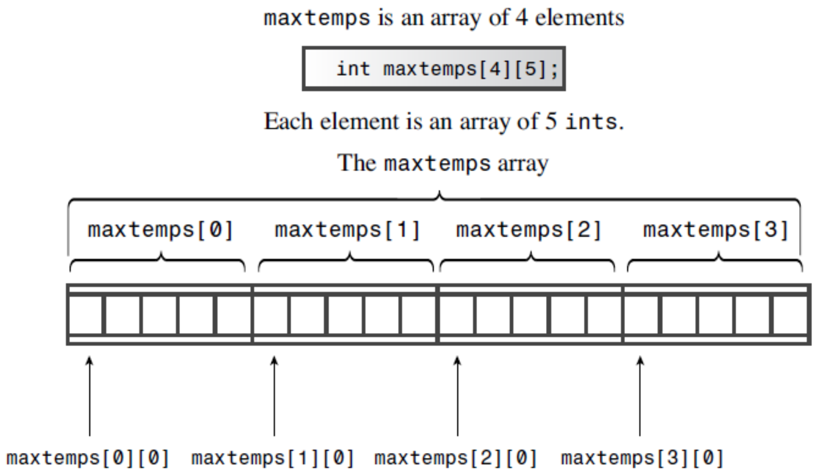
- Using unadorned cin for input
  - When to stop?
    - ✓ A sentinel character
  - Run textin1.cpp
    - ✓ The program omit the spaces
    - ✓ Program and operating system both work
- cin.get(char) to the rescue
  - Run textin2.cpp
    - ✓ Read the space
    - ✓ Declare the argument as a reference

Example: Nested Loops and Two-Dimensional Arrays

- Example:

int maxtemps[4][5];

- Run nested.cpp



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]

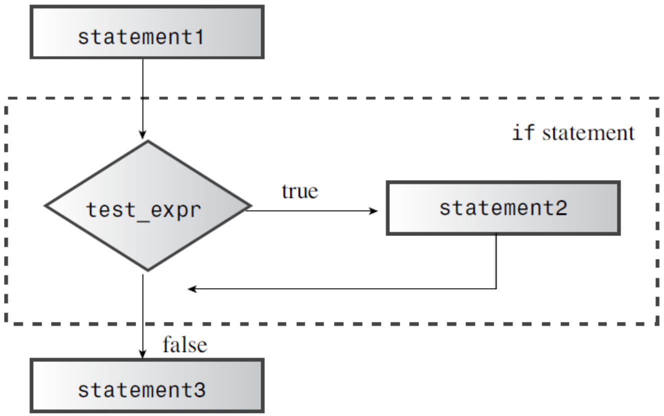
Branching Statements

The **if** Statement

- One of the keys to designing intelligent programs is to give them the ability to make decisions
  - ➢ Looping
  - ➢ if statement

- Run if.cpp

```
statement1
if (test_expr)
        statement2
statement3
```

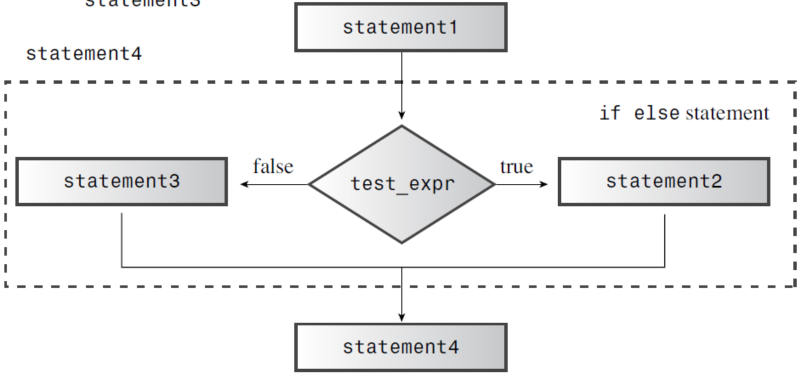


More than one selections

- The if else Statement
  - Decide which of two statements or blocks is executed
  - Must use braces to collect statements into a single block
  - Remember the conditional compilation #if, #else
- The if else if else Construction
- Run ifelseif.cpp

```
statement1
if (test_expr)
        statement2
else
        statement3

statement4
```



Logical Expressions

The Logical OR Operator: ||

- Three operators
  - Logical OR, written ||
  - Logical AND, written &&
  - Logical NOT, written !
- The logical OR operator: ||
  - || has a lower precedence than the relational operators
  - The || operator is a sequence point
  - C++ won't bother evaluating the expression on the right if the expression on the left is true

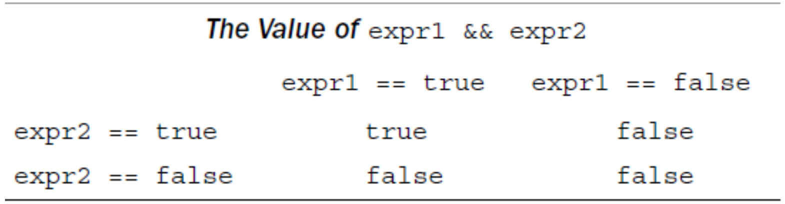
The Value of `expr1 || expr2`

	`expr1 == true`	`expr1 == false`
`expr2 == true`	`true`	`true`
`expr2 == false`	`true`	`false`

```
bool a = 1,b=1;
if (a||b++)
{
}
```

AND Operator: &&
NOT Operator: !

- AND Operator
  - Lower precedence than the relational operators
  - Acts as a sequence point
  - C++ doesn't bother evaluating the right side in some cases
- Run and.cpp
- NOT Operator
  - Exclamation point
  - If expression is true, or nonzero, then !expression is false
  - If expression is false, then !expression is true



The Value of `expr1 && expr2`

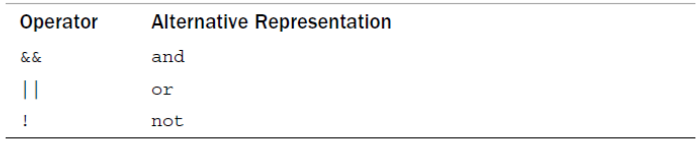
	`expr1 == true`	`expr1 == false`
`expr2 == true`	`true`	`false`
`expr2 == false`	`false`	`false`

Logical Operator Facts

- Precedence
  - The NOT(!) operator has a higher precedence than any of the relational or arithmetic operators
  - The AND operator has a higher precedence than the OR operator
  - Use parentheses to tell the program the interpretation you want

NOT-----relational-----AND-----OR

- Alternative Representations



Operator	Alternative Representation
`&&`	`and`
`||`	`or`
`!`	`not`

- The cctype library of character functions
  - A handy package of character-related functions

The ?: Operator

- Conditional operator (question mark)
  - More concise

```
int c;
if (a > b)
    c = a;
else
    c = b;
```

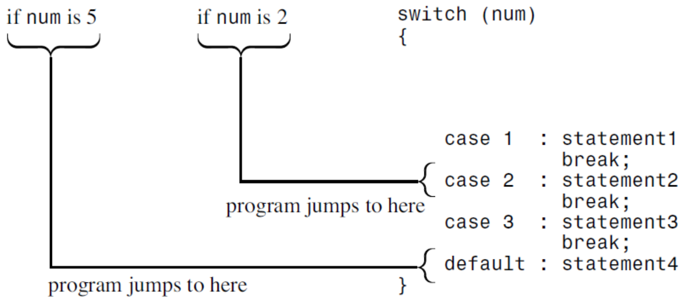
$\Downarrow$

```
int c = a > b ? a : b;
```

The switch Statement

- Acts as a routing device that tells the computer which line of code to execute next
- You must use the **break**

```
switch (integer-expression)
{
        case label1 : statement(s)
        case label2 : statement(s)
       ...
        default     : statement(s)
}
```



```
switch (num)
{
     case 1  : statement1
               break;
     case 2  : statement2
               break;
     case 3  : statement3
               break;
     default : statement4
}
```

- Run switch.cpp

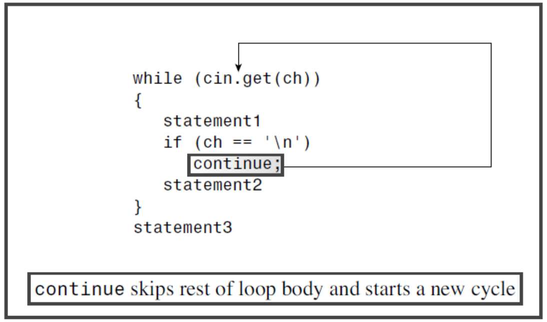
More About switch

- Using enumerators as labels
  - Run enum.cpp

- switch and if else
  - Let a program select from a list of alternatives
  - A switch statement isn't designed to handle ranges
  - Each switch case label must be a single value
  - Also that value must be an integer
  - A switch statement can't handle floating-point tests

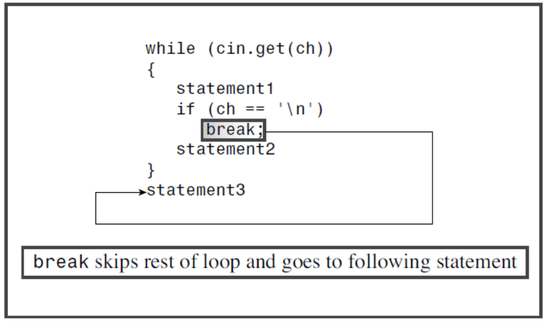
The break and continue Statements

- The break and continue statements enable a program to skip over parts of the code
  - break causes program execution to pass to the next statement following the switch or the loop
  - continue statement is used in loops and causes a program to skip the rest of the body of the loop and then start a new loop cycle
- Run jump.cpp



```
while (cin.get(ch))
{
    statement1
    if (ch == '\n')
      continue;
    statement2
}
statement3
```

continue skips rest of loop body and starts a new cycle



```
while (cin.get(ch))
{
    statement1
    if (ch == '\n')
      break;
    statement2
}
statement3
```

break skips rest of loop and goes to following statement

Example: Number-Reading Loops

- What happens if the user responds by entering a word instead of a number?

```
int n;
cin >> n;
```

- Run cinfish.cpp
  - ➢ The preceding example doesn't attempt to read any input after non-numeric input
- Run cingolf.cpp

Simple File Output

- Main steps for using file output
  - ➢ Include the fstream header file
  - ➢ Create an ofstream object
  - ➢ Associate the ofstream object with a file (C-style) using open()
  - ➢ Use the ofstream object in the same manner you would use cout
  - ➢ Use the close() method to close the file
- Run outfile.cpp

Simple File Input

- Main steps for using file input
  - Include the fstream header file and account for the std
  - Declare one or more ifstream variables, or objects
  - Associate a ifstream object with a file using open()
  - Use the close() method to close the file
  - Use >> operator, get(), getline(), …… method
- Run sumafile.cpp
  - What happens if you attempt to open a non-existent file for input?
  - exit(EXIT_FAILURE);
  - Communicate with the operating system
  - Terminate the program

Summary

- Loops
  - Increment/decrement operators: ++; --
  - Rational expressions: 6
  - for, while, do while
- Branch statements
  - if; if else; if else if else; switch
- The Logical Operator
  - OR, AND, NOT
- Jump operations
  - break and continue
- File fstream
  - Simple File Output: ofstream
  - Simple File Input: ifstream

CS 205 C/C++ Week 5

SID:	Name:	Q1:	Q2:	Q3:	Q4:	Q5:

Q1. int a[][4]={1,2,3,4,5,6,7,8,9,10}; int *p=*(a+1);
Which expression can get the value 9?
A. p+=3,*p++; B. p+=4,*(p++); C. p+=4,*++p; D. p+=4,++*p;
二维数组的名称是二级指针
Answer: A

Q 2. There are some declarations, which one can complete the assignment function of i=j?
int i, j=2, *p=&i;
A. *p=*&j; B. i=&j;
Answer: A

Q3. What's the output of the source code?

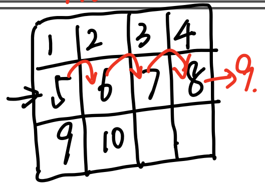
```
char *p = "abcdefgh", *r;
long *q = (long *)p;
q++;
r = (char *)q;
cout << r << endl;
```

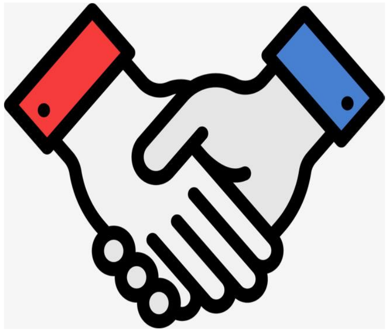
把 p 强转成 long pointer.
Answer: null

Q4. Define two variables： int a[5], *p; which description is wrong ?
A. p=p+1 B. $\boxed{\text{a=a+1}}$
Answer: B

Q5. char *name="newspaper"; The output of the cout<<name+2; is ( )
Answer: spaper.

Q1





Thanks

zhengf@sustech.edu.cn