哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

高级语言程序设计
High-level Language Programming

Lecture 5 Programming Control Structures

Yitian Shao (shaoyitian@hit.edu.cn)
School of Computer Science and Technology

Programming Control Structures

Course Overview

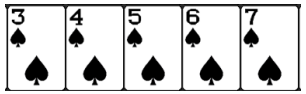
- The basics of algorithms
  - How to describe algorithms
- The building blocks of algorithms
  - Sequence
  - Selection
  - Iteration
- Rational operation
  - Logical expression and operators

Try it yourself on https://www.onlinegdb.com

Algorithms

- An **algorithm** describes **how to solve a problem**; it is a **procedure** that takes in **input**, follows a certain set of steps, and then produces an **output**

Example problem: Given a set of five cards (randomly shuffled), pick the **largest one**



Input: A set of 5 cards

Output: The card with the largest value

Procedure: **Up to the designer of the algorithm**

Algorithms

- Which **existing algorithms** to use?
- How to make **new algorithms** that are correct and efficient?

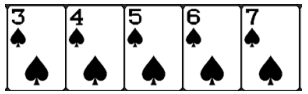
- How to **describe** an algorithm? (for others to understand)

The ways to describe an algorithm

- **Language description**: Might be in the form of text that describes the algorithm; generally does not involve implementation details of the algorithm.
- **Pseudocode**: Loosely formalizing an algorithm, with **general implementation details**; (programming) language-specific details are left out so as not to complicate things.
- **Flowchart**: Visual representation that depicts the step-by-step procedure of an algorithm; can help **simplify complex algorithms** into **visually understandable** forms.
- **Implementation**: An implementation in a given programming language will be a piece of code that is understandable and runnable by a computer; it will fulfill the goals and procedure of the algorithm.

The ways to describe an algorithm

- **Language description**: Might be in the form of text that describes the algorithm; generally does not involve implementation details of the algorithm.



Example of language description:

Input: A set of 5 cards

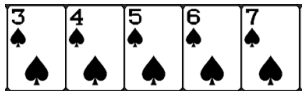
Output: The card with the largest value

Procedure:

Hold the first card in your hand and compare it with the remaining cards one by one. If the card you're holding has a smaller value, swap it with the card you're comparing it to.

The ways to describe an algorithm

- **Pseudocode**: Loosely formalizing an algorithm, with general implementation details; language-specific details are left out so as not to complicate things.



Example of pseudo-code:

Input: Given a set of card values $\{c_i\}$, with card index $i = 0,1,2,3,4$

Output: the largest value $\mathbf{res}$

Procedure:

Initialize $\mathbf{res} = c_0$

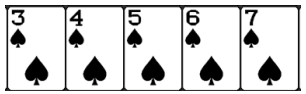
For each card index i form 1 to 4:
if $\mathbf{res} < c_i$, then let $\mathbf{res} = c_i$

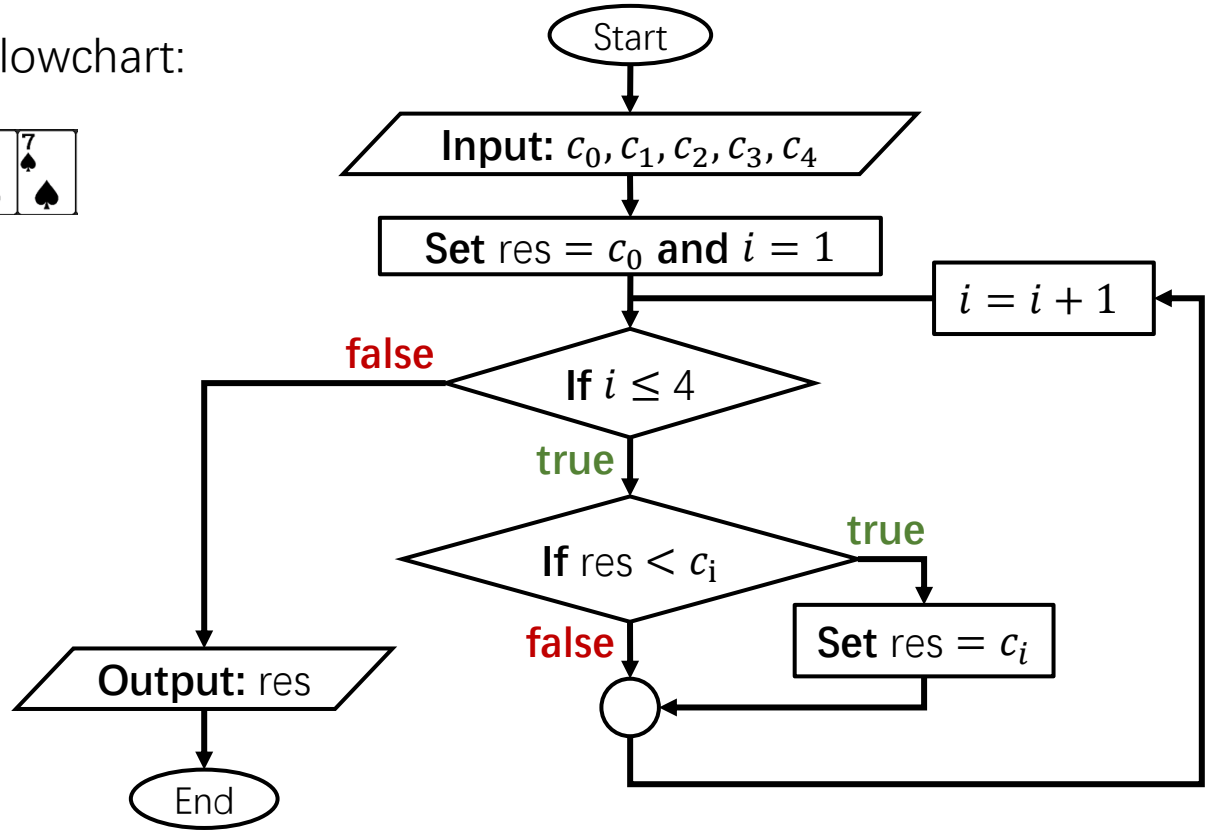
End

The ways to describe an algorithm

- **Flowchart**: Visual representation that depicts the step-by-step procedure of an algorithm; can help simplify complex algorithms into visually understandable forms.

Example of flowchart:

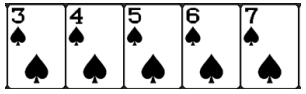




The ways to describe an algorithm

- **Implementation**: An implementation in a given programming language will be a piece of code that is understandable and runnable by a computer; it will fulfill the goals and procedure of the algorithm.

Example of C++ implementation:



main.cpp +

```cpp
#include<iostream>

int main()
{
    int c[5] = {5, 6, 7, 3, 4};

    int res = c[0];

    for(int i = 1; i <= 4; ++i)
    {
        if(res < c[i])
        {
            res = c[i];
        }
    }

    std::cout << res << std::endl;

    return 0;
}
```

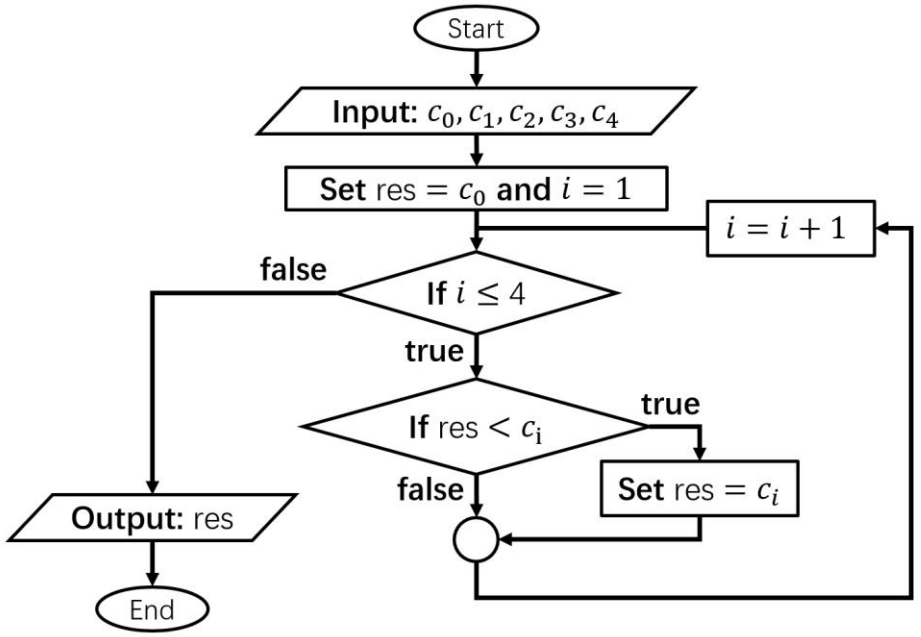
Ln: 21, Col: 1

Run | Share | Command Line Arguments

```
7
```

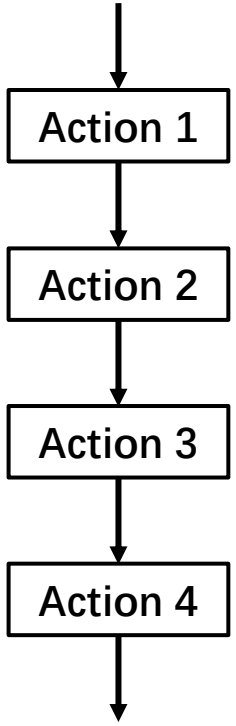
The building blocks of algorithms

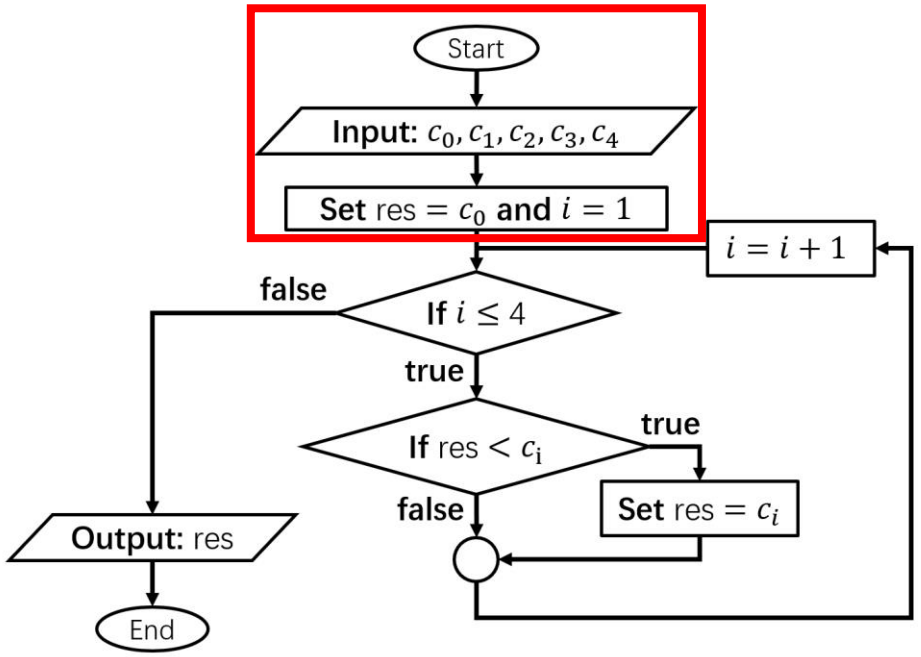
- An algorithm is made up of three basic **building blocks**:
  - Sequence
  - Selection
  - Iteration



Sequence

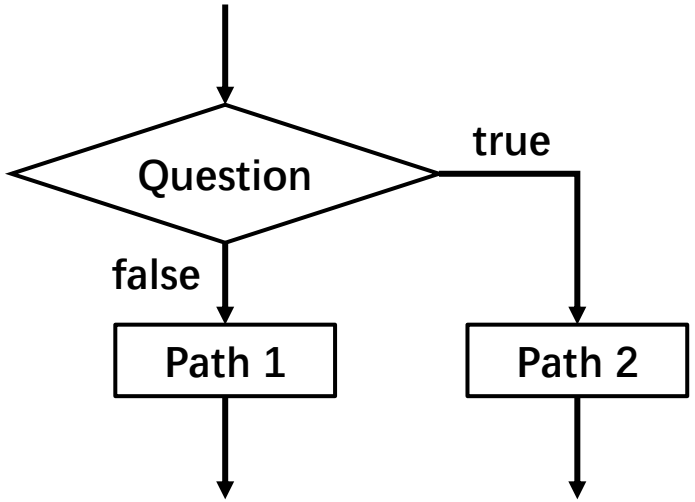
- A **sequence** is a series of actions (code statements) that is completed in a specific order, until all of the actions in the sequence have been carried out

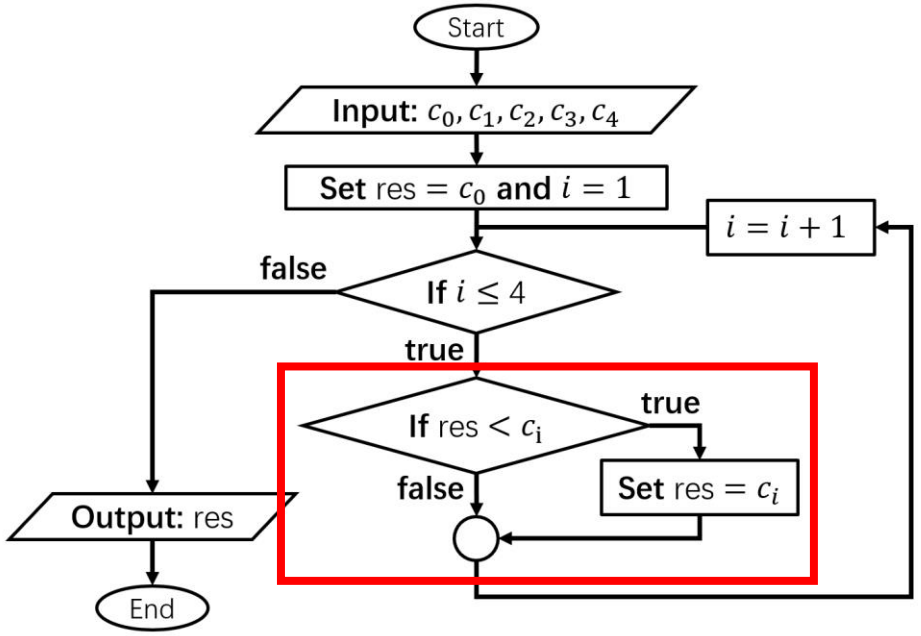




Selection

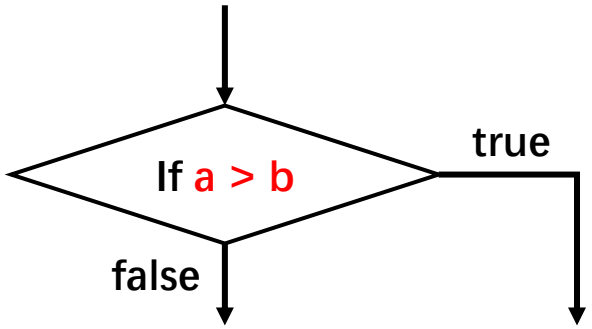
- Instead of following a specific order of actions, **selection** ask a question in order to figure out which path to take next.



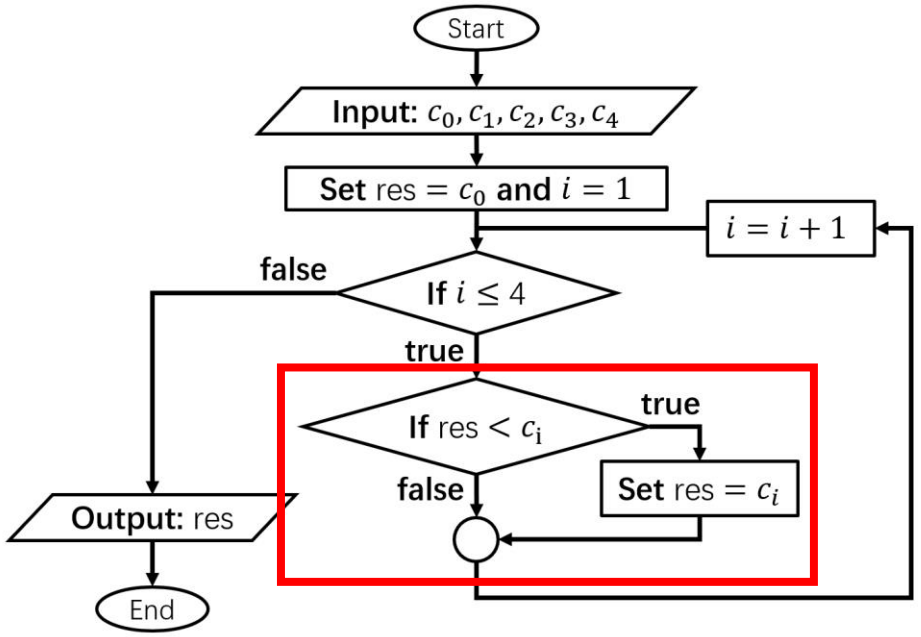


Selection

- Use **relational operation** for the selecting question




```
if(a > b)
{
    // Path 1
}
else
{
    // Path 2
}
```



Relational operation

- **Compare two** values or expressions; the comparison will return a boolean value (either **true** or **false**) as the result (= 1 or = 0)

operand1 **relational_operator** *operand2*

Example:

a > b

a <= b

a != b

Relational Operator	Meaning
>	Greater than
<	Less than
>=	Greater than equal to
<=	Less than equal to
==	Equal to
!=	Not equal to

Relational operation

- **Compare two** values or expressions; the comparison will return a boolean value (either **true** or **false**) as the result (= 1 or = 0)

operand1 **relational_operator** *operand2*

Example: a > b

expression1 **relational_operator** *expression2*

Example: (a + b) > (a < b)

(**An Expression** contains only identifiers, literals, and operators)

Relational Operator	Meaning
>	Greater than
<	Less than
>=	Greater than equal to
<=	Less than equal to
==	Equal to
!=	Not equal to

Relational operation

Relational Operator	Meaning	Examples	Result
>	Greater than	29 > 6	1 (true)
<	Less than	13 < 5	0 (false)
>=	Greater than equal to	5 >= 5	1 (true)
<=	Less than equal to	9 <= 12	1 (true)
==	Equal to	7 == 5	0 (false)
!=	Not equal to	7 != 5	1 (true)

No space, be sure to put '=' at the end: >= <= !=
A common mistake for new programmer: misuse of '=' and "=="

Relational operation

Relational Operator	Meaning
>	Greater than
<	Less than
>=	Greater than equal to
<=	Less than equal to
==	Equal to
!=	Not equal to

Precedence

5	a*b a/b a%b	Multiplication, division, and remainder
6	a+b a-b	Addition and subtraction
7	<< >>	Bitwise left shift and right shift
8	<=>	Three-way comparison operator (since C++20)
9	< <= > >=	For relational operators < and ≤ and > and ≥ respectively
10	== !=	For equality operators = and ≠ respectively
11	a&b	Bitwise AND
12	^	Bitwise XOR (exclusive or)
13	|	Bitwise OR (inclusive or)
14	&&	Logical AND
15	||	Logical OR

Example: 2 <= 4 – 3

Relational operation

Relational Operator	Meaning
>	Greater than
<	Less than
>=	Greater than equal to
<=	Less than equal to
==	Equal to
!=	Not equal to

Precedence

5	a*b a/b a%b	Multiplication, division, and remainder
6	a+b a-b	Addition and subtraction
7	<< >>	Bitwise left shift and right shift
8	<=>	Three-way comparison operator (since C++20)
9	< <= > >=	For relational operators < and ≤ and > and ≥ respectively
10	== !=	For equality operators = and ≠ respectively
11	a&b	Bitwise AND
12	^	Bitwise XOR (exclusive or)
13	|	Bitwise OR (inclusive or)
14	&&	Logical AND
15	||	Logical OR

Example: 2 <= 4 – 3

2 <= (4 – 3)

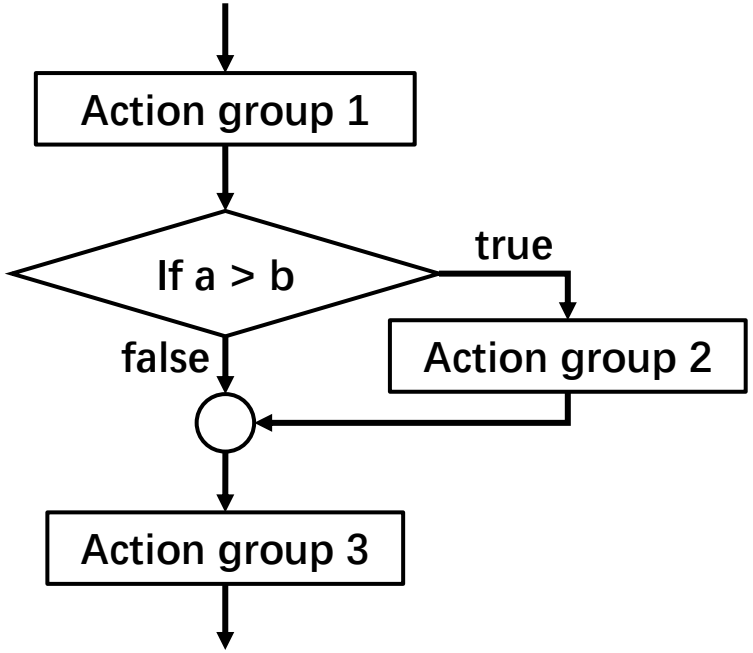
2 <= 1

false (0)

Single Selection

- Single Selection

```
// Action group 1
if(a > b)
{
    // Action group 2
}
// Action group 3
```



Action: one line of code statement
Action group: multiple lines of code statement

Single Selection

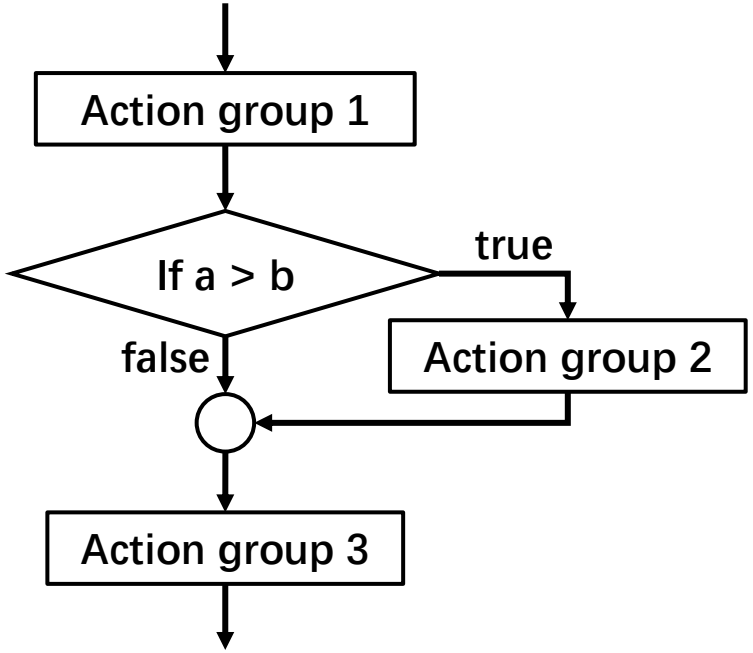
- Single Selection

```
if(a > b)
{
    c = c + 1;
    d = 2 * c;
    // and more actions
}
```

Use curly brackets

```
if(a > b){
    c = c + 1;
    d = 2 * c;
    // and more actions
}
```

Different coding style



Single Selection

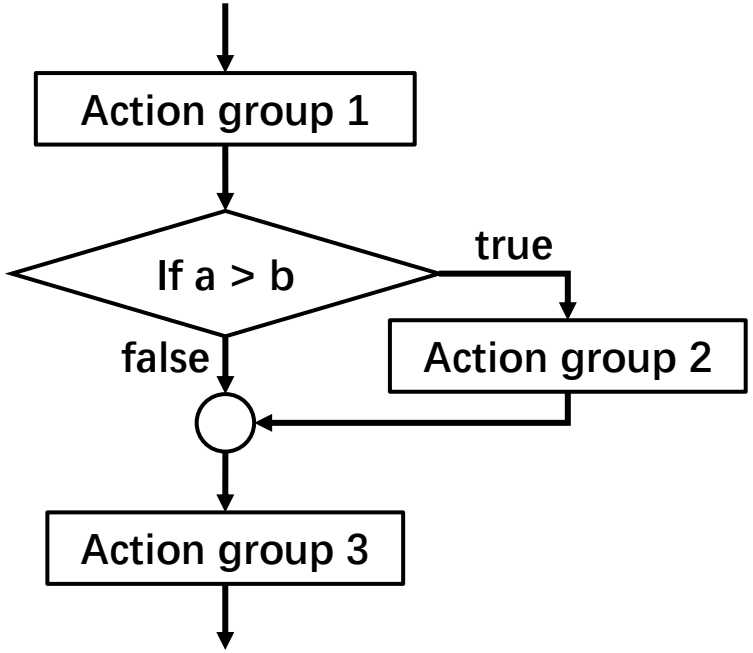
- Single Selection

```
if(a > b)
{
    c = c + 1;
    d = 2 * c;
    // and more actions
}
```

```
if(a > b){
    c = c + 1;
    d = 2 * c;
    // and more actions
}
```

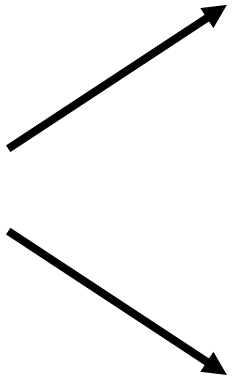
```
if(a > b) c = c + 1;
```

Without curly brackets, one action (code statement) only, not suggested for beginners



Single Selection

if (***expression***)
{
    actions
}



if (Boolean value)
{
    actions
}

true / false

if (numerical value)
{
    actions
}

0 = false, otherwise = true

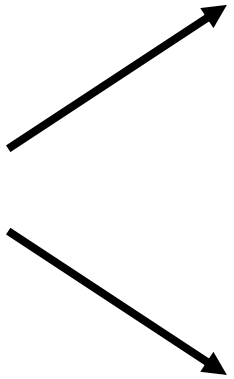
```
int c = 0;

if(100) c=1;
```

```
int c = 0;

if(0.0001) c=1;
```

c = ?

Single Selection

```
if (expression)
{
        actions
}
```



```
if (Boolean value)
{
        actions
}
```

true / false

```
if (numerical value)
{
        actions
}
```

0 = false, otherwise = true

```
int c = 0;

if(100) c=1;
```

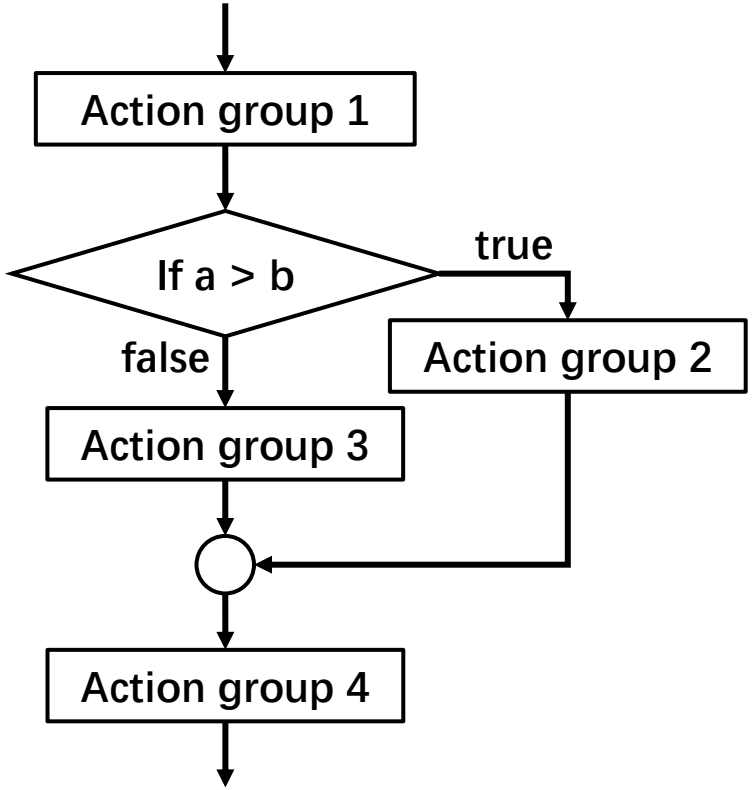
```
int c = 0;

if(0.0001) c=1;
```

c = 1 in both cases

Double Selection

- Double Selection

```
// Action group 1
if(a > b)
{
    // Action group 2
}
else
{
    // Action group 3
}
// Action group 4
```




```
if(a > b) c = 1;
else c = 2;
```

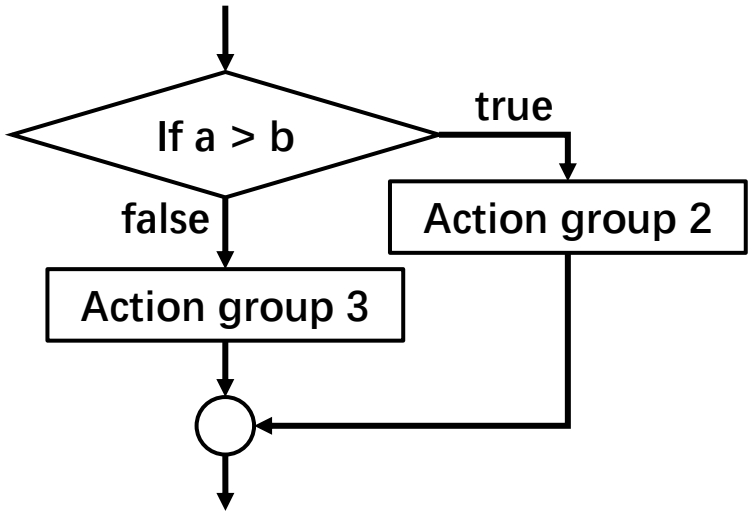
Valid, but not suggested

Double Selection

Which is more efficient?

```
if(a > b)
{
    // Action group 2
}
else
{
    // Action group 3
}
```

```
if(a > b)
{
    // Action group 2
}
if(a <= b)
{
    // Action group 3
}
```



Conditional operator

- Shortest form of writing conditional statements
- Can use as inline conditional statement in place of if-else statement
- Syntax:

expression ? statement_1 : statement_2

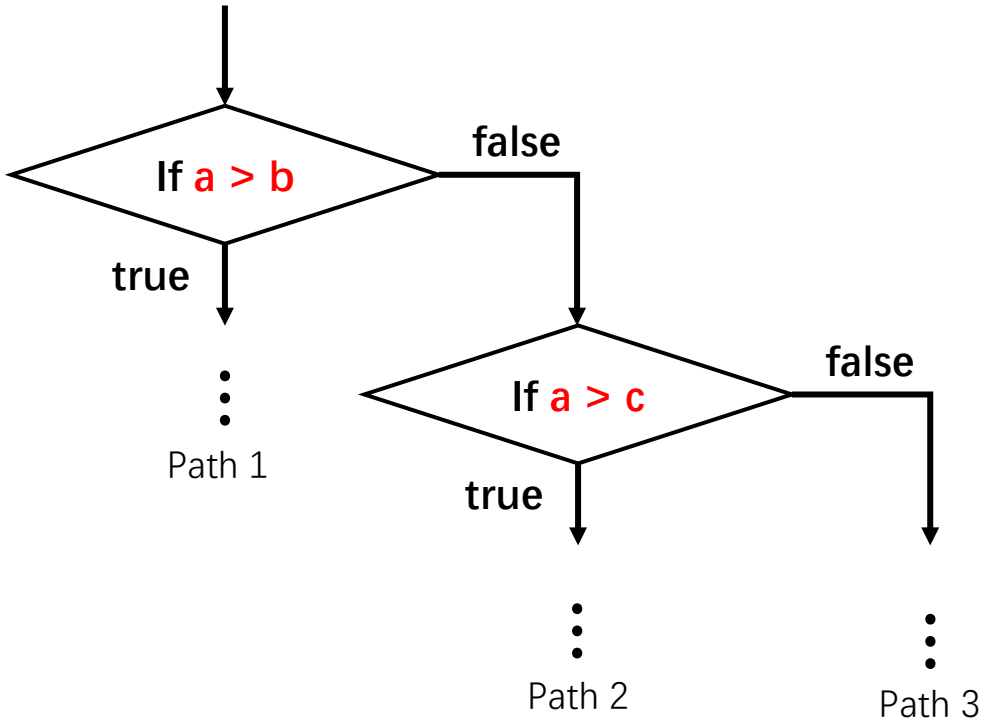
Example

```
max_value = a > b ? a : b;
```

is equivalent to

```
if(a > b)
{
    max_value = a;
}
else
{
    max_value = b;
}
```

Multiple selection

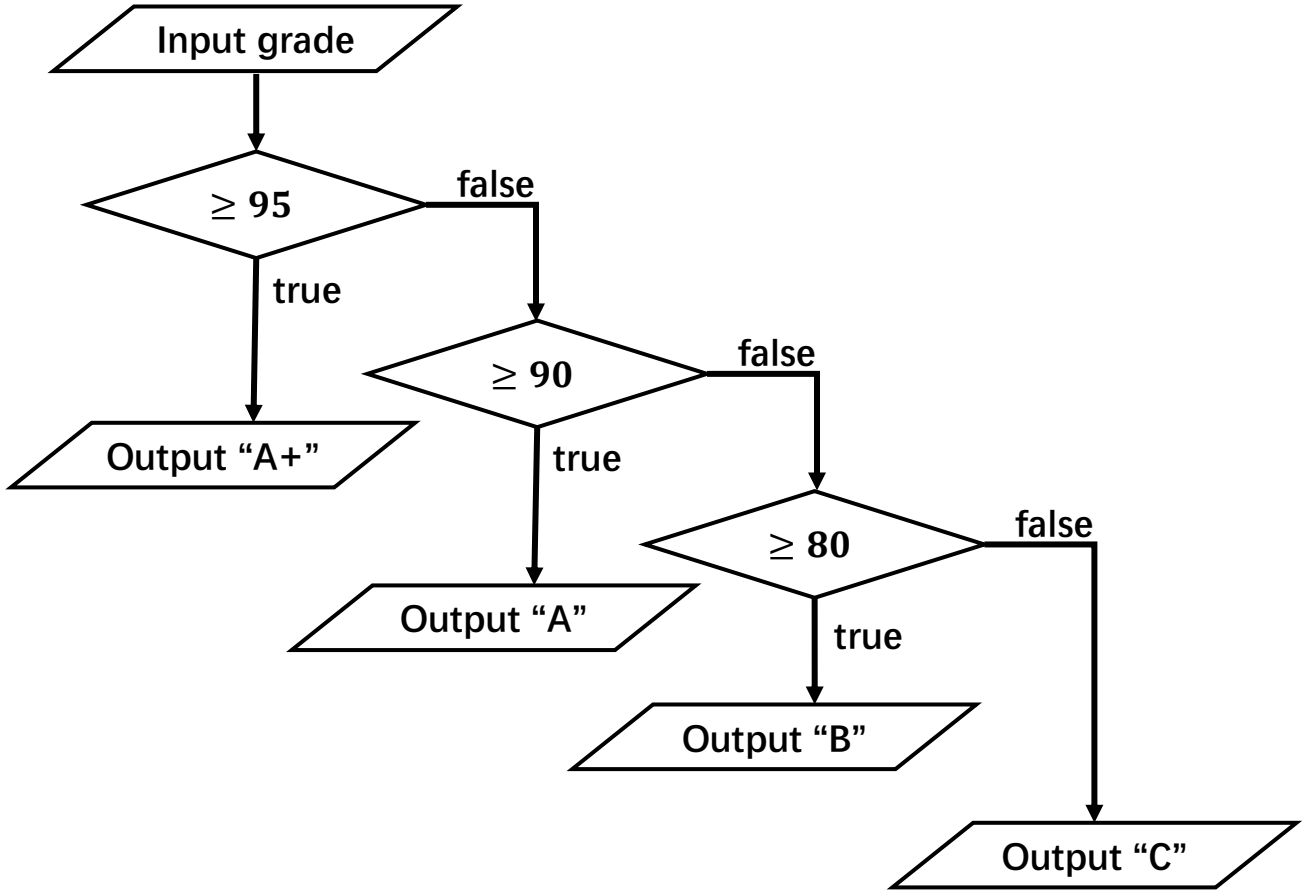



```
if(a > b)
{
    // Path 1
}
else if(a > c)
{
    // Path 2
}
else
{
    // Path 3
}
```

Can add multiple "**else if**", but one "**else**" only

Multiple selection

Example: a simple grade conversion system




```
int grade;
std::cin >> grade;

if(grade >= 95)
{
    std::cout << "A+"
}
else if(grade >= 90)
{
    std::cout << "A";
}
else if(grade >= 80)
{
    std::cout << "B";
}
else
{
    std::cout << "C";
}
```

Multiple selection: switch statements

- A **switch** is a control statement in which the control of the program depends on **the value passed to switch function**. When a match is found, the statements associated with that constant are executed.

- Syntax

```
switch( expression )
{
        case  value_1:
              {Action group 1}
              break;
        case  value_2:
              {Action group 2}
              break;
              ⋮
        default:
              {Action group n}
              break;
}
```

The result of **expression** must be of **int** or **char** type

The value must be a **constant**, and don't forget the colon

Multiple selection: switch statements

Example

main.cpp

```cpp
#include<iostream>

int main()
{
    int month = 2;

    switch(month)
    {
        case 1:
        std::cout << "Jan";
        break;
        case 2:
        std::cout << "Feb";
        break;
        case 3:
        std::cout << "Mar";
        break;
        default:
        std::cout << month;
        break;
    }

    return 0;
}
```

Ln: 25, Col: 1

Run | Share | Command Line Argumer

Feb

main.cpp

```cpp
#include<iostream>

int main()
{
    int month = 3;

    switch(month)
    {
        case 1:
        std::cout << "Jan";
        break;
        case 2:
        std::cout << "Feb";
        break;
        case 3:
        std::cout << "Mar";
        break;
        default:
        std::cout << month;
        break;
    }

    return 0;
}
```

Ln: 25, Col: 1

Run | Share | Command Line Argu

Mar

main.cpp

```cpp
#include<iostream>

int main()
{
    int month = 12;

    switch(month)
    {
        case 1:
        std::cout << "Jan";
        break;
        case 2:
        std::cout << "Feb";
        break;
        case 3:
        std::cout << "Mar";
        break;
        default:
        std::cout << month;
        break;
    }

    return 0;
}
```

Ln: 25, Col: 1

Run | Share | Command Line Argum

12

Multiple selection: switch statements

Example

main.cpp

```cpp
#include<iostream>

int main()
{
    int month = 2;

    switch(month)
    {
        case 1:
        std::cout << "Jan";
        break;
        case 2:
        std::cout << "Feb";
        break;
        case 3:
        std::cout << "Mar";
        break;
        default:
        std::cout << month;
        break;
    }

    return 0;
}
```

Ln: 25, Col: 1

Run | Share | Command Line Argumer

Feb

Equivalent implementation using if-else statements

main.cpp

```cpp
#include<iostream>

int main()
{
    int month = 2;

    if(month == 1)
    {
        std::cout << "Jan";
    }
    else if(month == 2)
    {
        std::cout << "Feb";
    }
    else if(month == 3)
    {
        std::cout << "Mar";
    }
    else
    {
        std::cout << month;
    }

    return 0;
}
```

Ln: 26, Col: 1

Run | Share | Command Line Argume

Feb

Multiple selection: switch statements

What if we remove the “break”?

main.cpp

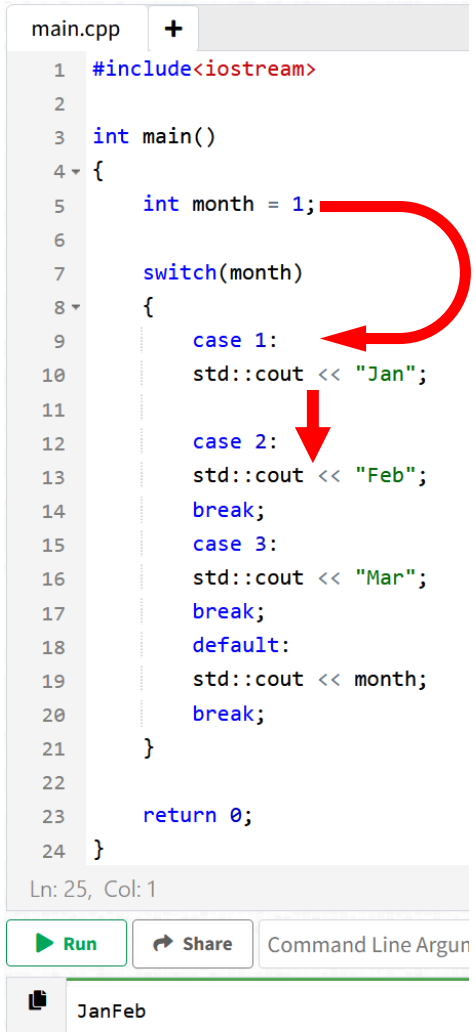
```cpp
#include<iostream>

int main()
{
    int month = 1;

    switch(month)
    {
        case 1:
        std::cout << "Jan";

        case 2:
        std::cout << "Feb";
        break;
        case 3:
        std::cout << "Mar";
        break;
        default:
        std::cout << month;
        break;
    }

    return 0;
}
```

Ln: 25, Col: 1

Run | Share | Command Line Argun

Multiple selection: switch statements

What if we remove the “break”?



Logical expression and operators

- So far, we use only a single expression

```
// Action group 1
if(a > b)
{
    // Action group 2
}
// Action group 3
```

- What about conditioning on multiple expressions?

Example: want to search movies of 1990's (between 1990 and 2000)

```
if(year < 2000)
{
    if(year >= 1990)
    {
        std::cout << "1990's";
    }
}
```

Inconvenient!

Logical expression and operators

- **Logical operators** are symbols that can combine or modify conditions to make logical evaluations.

- Syntax

operand_a **logical_operator** *operand_b*

Symbol	Description
&&	AND: **true** only if both operands are **true**
||	OR: **true** if any of the operands is **true**
!	NOT: reverse the logic

logical_operator *single_operand*

a	*b*	*a* **&&** *b*	*a* **||** *b*	**!***a*	**!***b*
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

Logical expression and operators

Example: Search movies of 1990's

```
if(year >= 1990 && year < 2000)
{
    std::cout << "1990's";
}
```

(year >= 1990) && (year < 2000)

Precedence	Operator	Description
1	::	Scope resolution
2	a++ a--	Suffix/postfix increment and decrement
	type() *type*{}	Functional cast
	a()	Function call
	a[]	Subscript
	. ->	Member access
3	++a --a	Prefix increment and decrement
	+a -a	Unary plus and minus
	! ~	Logical NOT and bitwise NOT
	(*type*)	C-style cast
	*a	Indirection (dereference)
	&a	Address-of
	sizeof	Size-of[note 1]
	co_await	await-expression (C++20)
	new new[]	Dynamic memory allocation
	delete delete[]	Dynamic memory deallocation
4	.* ->*	Pointer-to-member
5	a*b a/b a%b	Multiplication, division, and remainder
6	a+b a-b	Addition and subtraction
7	<< >>	Bitwise left shift and right shift
8	<=>	Three-way comparison operator (since C++20)
9	< <= > >=	For relational operators < and ≤ and > and ≥ respectively
10	== !=	For equality operators = and ≠ respectively
11	a&b	Bitwise AND
12	^	Bitwise XOR (exclusive or)
13	|	Bitwise OR (inclusive or)
14	&&	Logical AND
15	||	Logical OR
16	a?b:c	Ternary conditional[note 2]
	throw	throw operator
	co_yield	yield-expression (C++20)
	=	Direct assignment (provided by default for C++ classes)
	+= -=	Compound assignment by sum and difference
	*= /= %=	Compound assignment by product, quotient, and remainder
	<<= >>=	Compound assignment by bitwise left shift and right shift
	&= ^= |=	Compound assignment by bitwise AND, XOR, and OR
17	,	Comma

Logical expression and operators

Example

```
int a = 3, b = 2, c = 1;

std::cout << (a > b > c);
```

Output ?

Logical expression and operators

Example

```
int a = 3, b = 2, c = 1;

std::cout << (a > b > c);
```

= (a > b) > c
= 1 > c
= 0

```
int a = 3, b = 2, c = 1;

std::cout << (a > b && b > c);
```

= (a > b) && (b > c)
= 1 && 1
= 1

Iteration

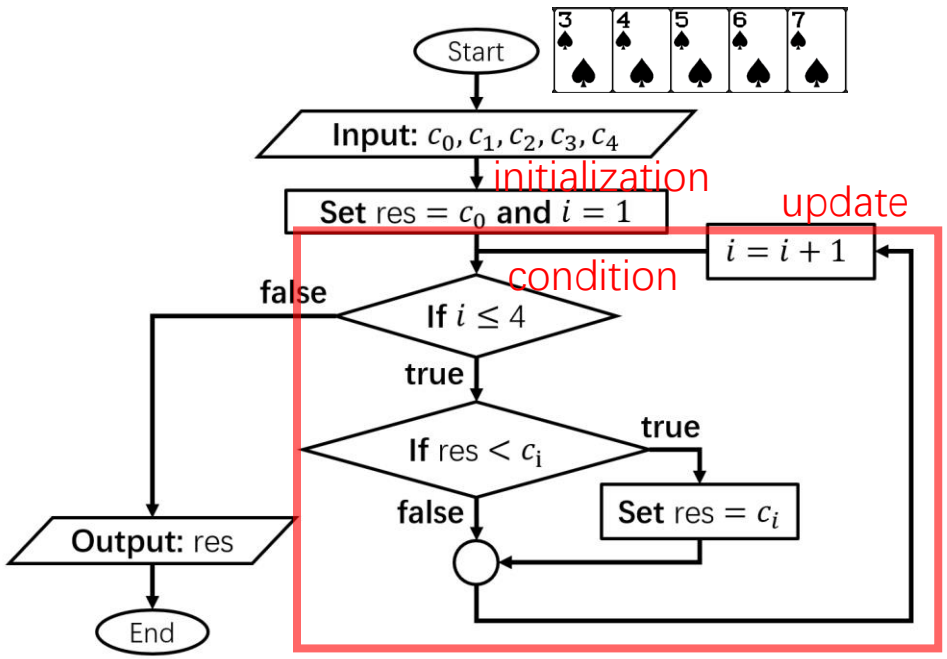
- In computer programming, **loops** are used to repeat a block of code
- For example, to show a message 100 times, instead of writing the cout statement 100 times, we can use a loop structure
- Three types of loops in C++
  - **for** loop
  - **while** loop
  - **do-while** loop

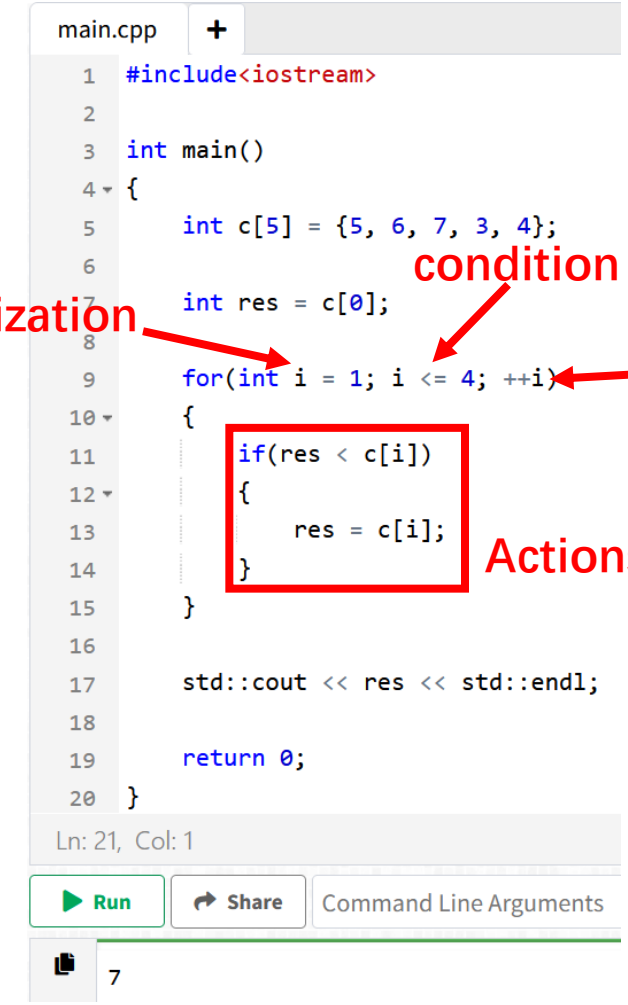
“for” loop

- Syntax

Use semicolons!

```
for (initialization ; condition ; update )
{
        Actions
}
```





“while” loop

- Syntax

while (**condition**)
{
    Actions
}

```
// custom countdown using while
#include <iostream>
using namespace std;

int main ()
{
  int n = 10;

  while (n>0) {
    cout << n << ", ";
    --n;
  }

  cout << "liftoff!\n";
}
```

condition

Actions

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, liftoff!
```

“do-while” loop

- Syntax

```
do
{
      Actions
}
while (condition)
```

Compared to the while loop structure

```
while (condition)
{
      Actions
}
```

```
#include <iostream>
using namespace std;

int main()
{
    float num, total;

    total = 0;
    num = 1;

    do
    {
        cout << "Please enter a number" ;
        cin >> num;
        total += num;
        cout << "The running total is" << total << endl << endl;
    }
    while(num != 0);

    cout << "The final total is" << total << endl;
}
```

Actions

condition

HOMEWORK

Homework 5

- 1. Rewrite the following if-else using a switch statement:

```
if ( marriage_status == 'S' )
  cout << "single" ;
else if ( marriage_status == 'M' )
  cout << "married" ;
else if ( marriage_status == 'W' )
  cout << "widowed" ;
else if ( marriage_status == 'E' )
  cout << "separated"  ;
else if ( marriage_status == 'D' )
  cout << "divorced"  ;
else
  cout << "error: invalid code" ;
```

Homework 5

- 2. In a triangle, the sum of any two sides must be greater than the third side. Write a program to input three numbers and determine if they form a valid triangle.
- 3. Rewrite the following using a for loop.

```
int i = 0, total = 0 ;
while ( i < 10 )
{
   cin >> n  ;
   total += n ;
   i++ ;
}
```

Homework 5

- 4. What is displayed when the following program is run and the number 1234 is entered?

```
int num ;
cout << "Please enter a number "  ;
cin >> num ;
do
{
  cout << num % 10 ;
  num /= 10 ;
}
while ( num != 0 ) ;
```

- 5. Write a program to find the sum of all the odd integers in the range 1 to 99.

Homework 5

- 6. Write a program to display the following triangles:

```
*
**
***
****

   *
  **
 ***
****
```

Hint:

*	*space*	*space*	*space*
*	*	*space*	*space*
*	*	*	*space*
*	*	*	*

（Bonus） Modify the size of the triangles (number of stars) by taking the keyboard input from a user. The example above has 4 stars in the last row.