C++ Programming

- Decision making

With Thanks to Dr. Xin Feng

Outline

- C/C++ statements
- Decision making statements
- `if`
- `if-else`
- `if-else-if`
- `switch`

C/C++ Statements

- A statement is a part of a program that can be executed.
- An expression can be a statement (simple statement).

```
a=a+1;
a--;
```

- A function call is also a statement (more about function call will be introduced later).
- A compound statement consists of several expressions and statements
  - Decision-making statements
  - Looping statements

C/C++ Statements

```
a=
a+
1;
```

Is this a statement?

C/C++ Statements

```
a=
a+
1;
```

It is a statement, but not recommended.

```
a=a+1;
```

This looks better.

Decision-Making Statements

- A decision-making statement allows us to control whether a program segment is executed or not.
- Two constructs
  - `if` statement
    - `if`
    - `if-else`
    - `if-else-if`
  - `switch` statement

An Example

```
 if (it is sunny){
         go to beach;
         swim;
  }
  else go to library;
```

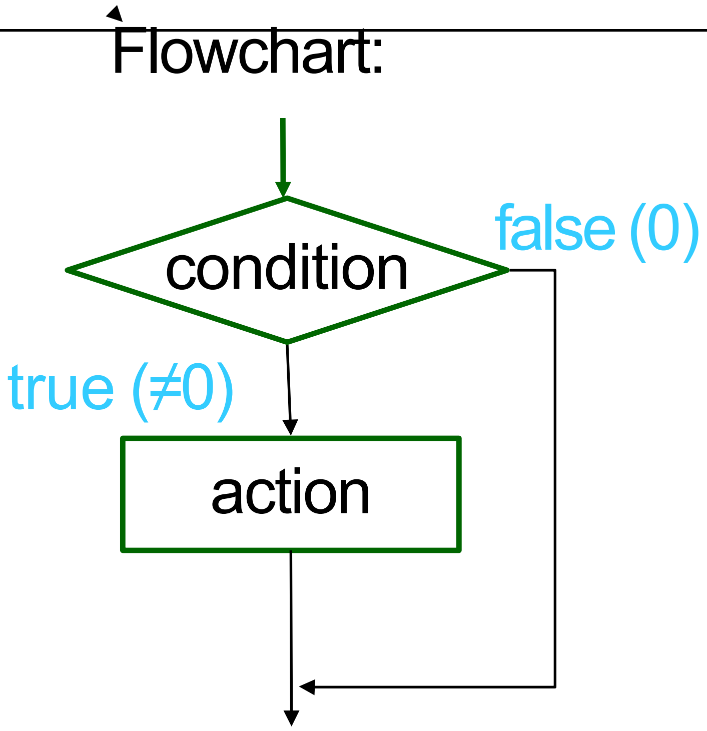
What does this mean?

The Basic `If` Statement

Syntax:

```
if(condition){
    action
}
```

Flowchart:



- If the condition is true then execute the action.
- action is either a single statement or a group of statements within curly brackets.

An Example

```
/* program to read number and print out its absolute value */
#include <iostream>
using namespace std;
int main()
{
    int value;
    cout << “Please enter an integer:";
    cin >> value ;
    if(value < 0)
       value = -value;
   cout << "The absolute value is " << value << endl;
    return 0;
}
```

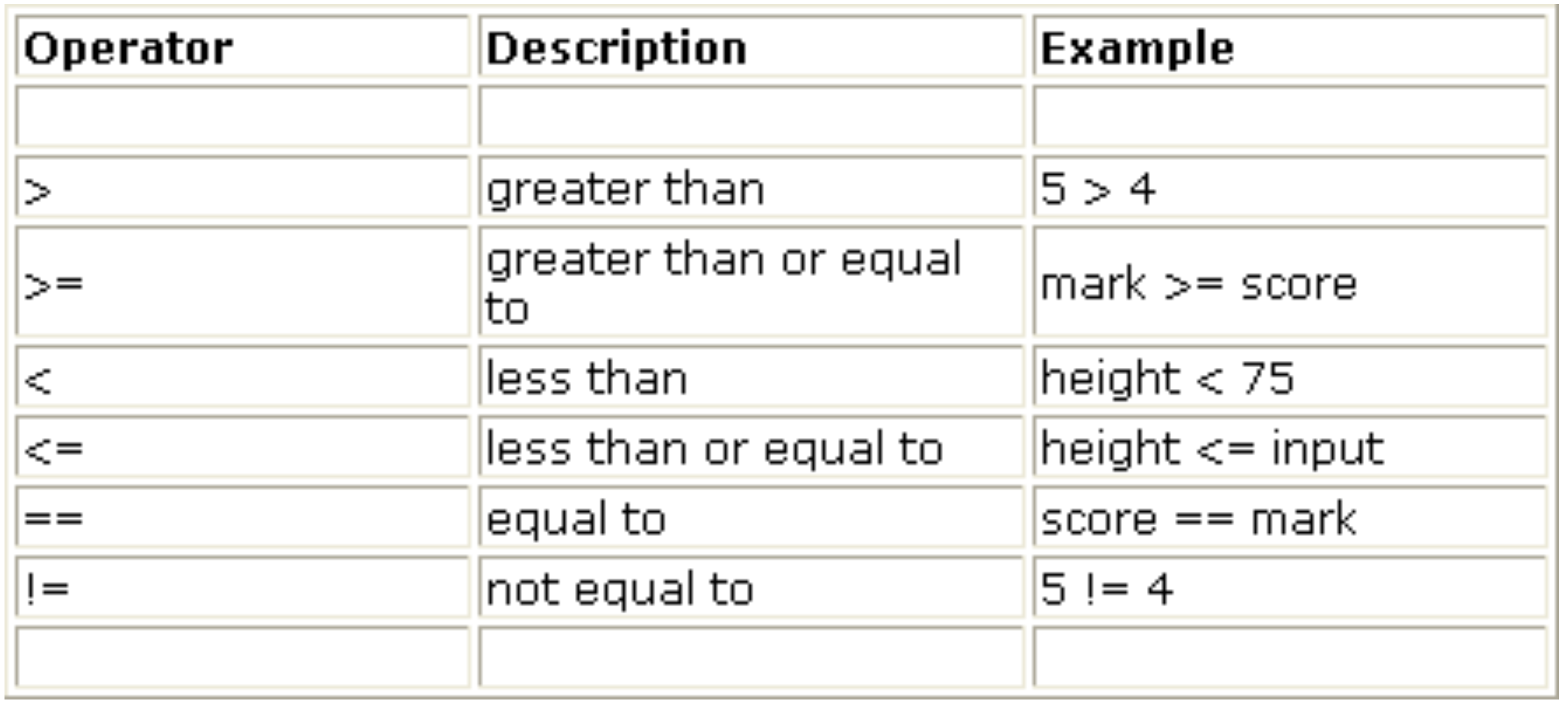
This program is to ???

An Example

```
/* program to read number and print out its absolute value */
#include <iostream>
using namespace std;
int main()
{
    int value;
    cout << "Please enter an integer:";
    cin >> value ;
    if(value < 0){
      value = -value;
     cout << "The absolute value is " << value << endl;
    }
   return 0;
}
```

What if a pair { } are added in this way?

Relational Expressions



Operator	Description	Example
>	greater than	5 > 4
>=	greater than or equal to	mark >= score
<	less than	height < 75
<=	less than or equal to	height <= input
==	equal to	score == mark
!=	not equal to	5 != 4

'=' and '=='

- Compare these two program segments

```
int a;
cin >> a;
if (a == 10)
   cout << "a is " << 10;
```

```
int a;
cin >> a;
if (a = 10)
  cout << "a is " << 10;
```

If input `5,10`, outputs are different or not?

'=' and '=='

- Compare these two program segments

```
int a;
cin >> a;
if (a == 10)
   cout << "a is " << 10;
```

```
int a;
cin >> a;
if (a = 10)
   cout << "a is " << 10;
```

Input: 5
Output:

Input: 10
Output: a is 10

Input: 5
Output: a is 10

Input: 10
Output: a is 10

'=' and '=='

- Compare these two program segments

```
int a;
cin >> a;
if (a == 10)
   cout << "a is " << 10;
```

```
int a;
cin >> a;
if (a = 10)
  cout << "a is " << 10;
```

Input: 0

Output: ???

Condition

- a condition can have one of two values:
  - `true` (corresponds to a non-zero value)
    - e.g., `if (x = 10)`, `if (10)`
  - `false` (corresponds to zero value)
    - e.g., `if (0)`

Condition

- The Boolean data type `bool`
- A `bool` variable stores only a `0` or `1`

```
int i = 7;
bool b1, b2;
b1 = 0;
b2 = i;
cout << "b1=" << b1 << ',' << "b2=" << b2;
```

Output: ???

Logical Operators

- Remember these logical operators?
  - && (and)
  - || (or)
  - ! (not)

What are the values of these Boolean variables

```
bool P = 1;
bool Q = 0;
bool R = 1;
bool S = P && Q;
bool T = !Q || R;
bool U = !(R && !Q);
```

Precedence of Operators

- Precedence of operators (from highest to lowest)

– Parentheses	( )
– Unary operators	!
– Multiplicative operators	*, /, %
– Additive operators	+, -
– Relational ordering	<, <=, >=, >
– Relational equality	== !=
– Logical and	&&
– Logical or	||
– Assignment	=

An Example

```
int a;
cin >> a;
if (a <= 10 && a >= 5)
   cout << "a is between 5 and 10";
```

What does this program segment do???

An Example

Sorting two numbers:

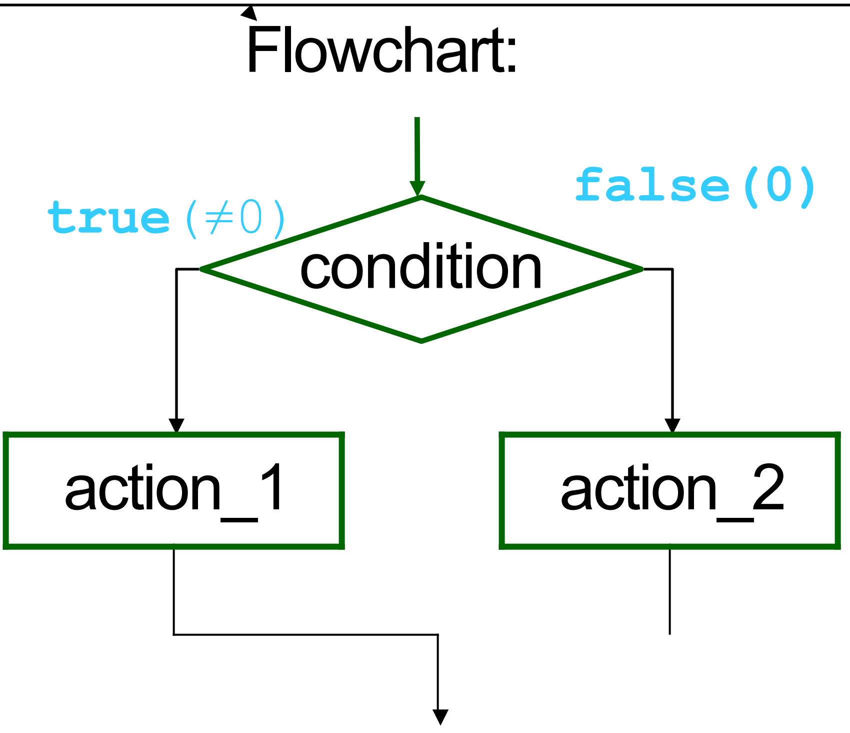
```
int value1;
int value2;
int temp;
cout << "Enter two integers:";
cin >> value1 >> value2;
if(value1 > value2){
   temp = value1;
   value1 = value2;
   value2 = temp;
}
cout << "The input in a sorted order: " ;
cout << value1 << value2  << endl;
```

The Basic `if - else` Statement

Syntax:

```
if(condition){
    action_1
}
else {
    action_2
}
```

Flowchart:



- If the condition is true then execute `action_1`; otherwise, execute `action_2`.
- `action_1` and `action_2` are either a single statement or a group of statements within curly brackets.

An Example

```
if (it is sunny){
   go to beach;
    swim;
 }
go to library;
```

```
if (it is sunny){
   go to beach;
   swim;
  }
else
   go to library;
```

Any difference???

An Example

```
int value1;
int value2;
int larger;
cout << "Enter two integers: ";
cin >> value1 >> value2;
if(value1 > value2)
   larger = value1;
else
   larger = value2;
cout << "Larger of inputs is: " << larger << endl;
```

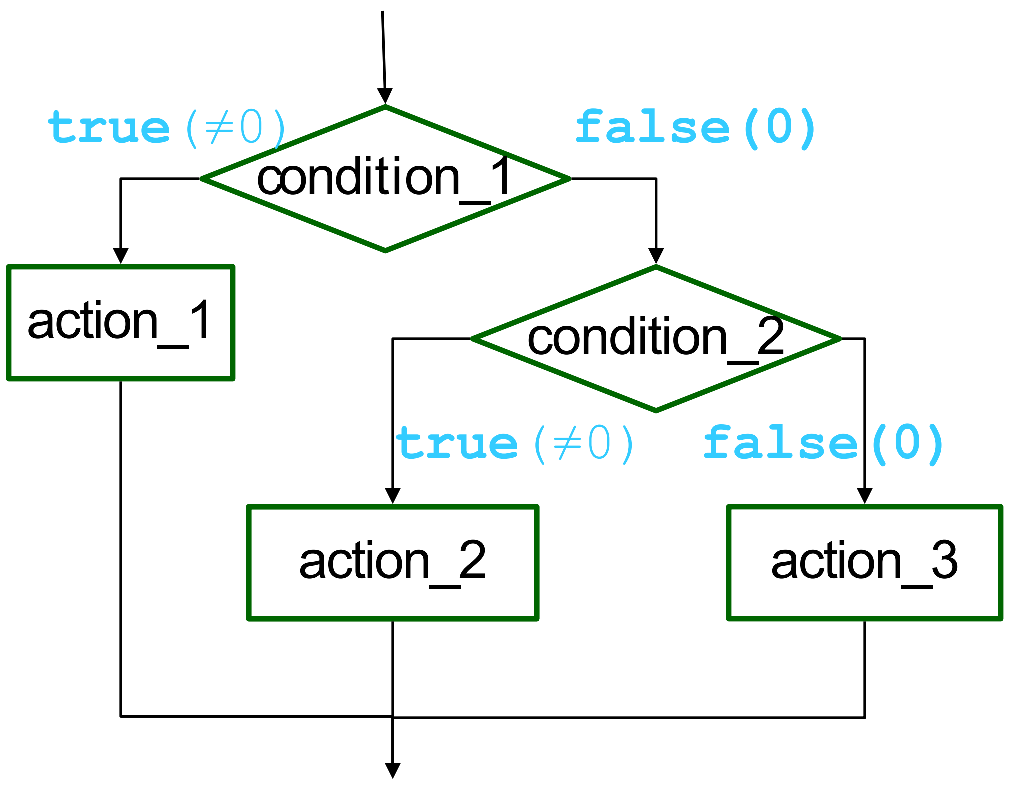
Input: 5 10   Output: ??

The Basic `if - else if` Statement

Syntax:

```
if(condition_1){
    action_1
}
else if(condition_2){
    action_2
}
else {
    action_3
}
```

Flowchart:



An Example

An integer calculator (for only +, − * and /):

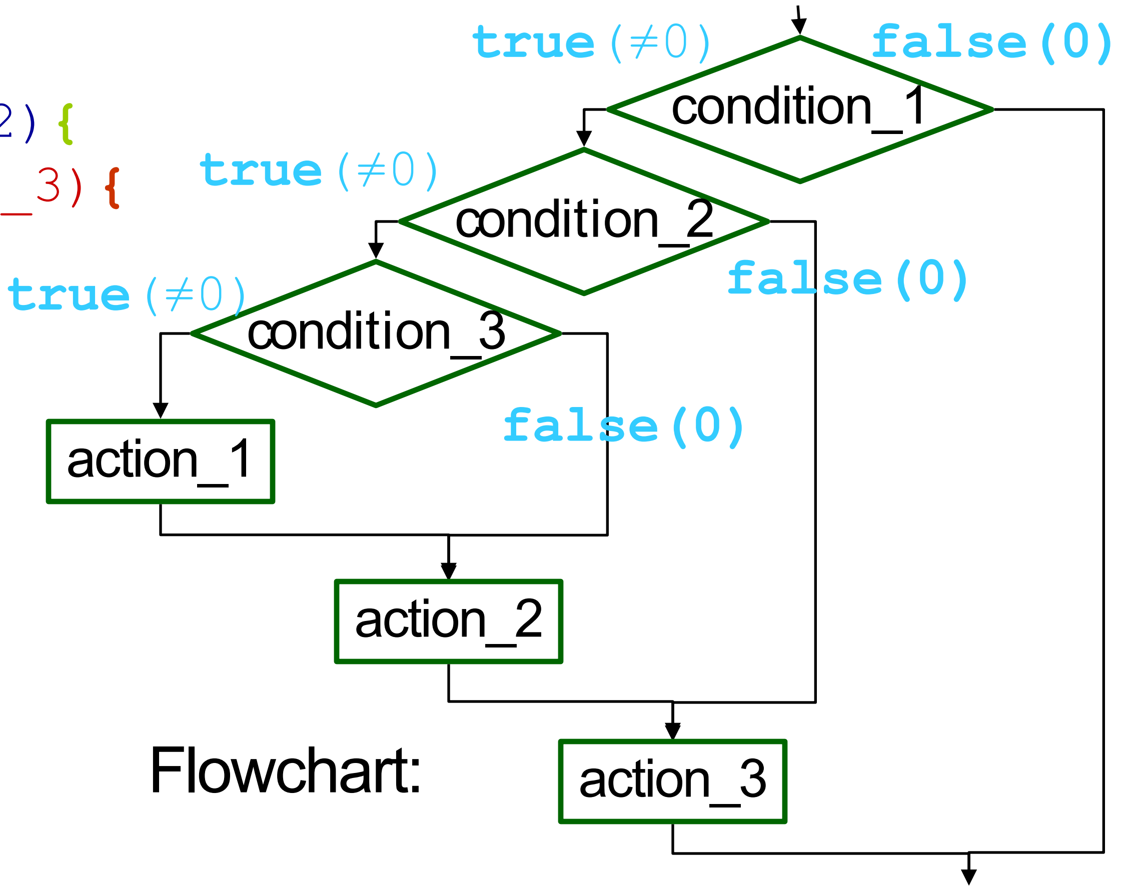
```
char op;
int x, y;
cin >> x >> op >> y;
if(op == '+')
    cout << x << '+' << y << '=' << x + y;  else
if(op == '-')
    cout << x << '-' << y << '=' << x - y;  else if(op
== '*')
    cout << x << '*' << y << '=' << x * y;  else
if(op == '/')
    cout << x << '/' << y << '=' << x / y;  else cout
<<                        "Invalid operator!";
```

Flowchart ???

Nested `if` Statements

- Nested means that one compound statement is inside another

```
if (condition_1){
     if (condition_2){
       if (condition_3){
            action_1
       }
       action_2
     }
     action_3
}
```

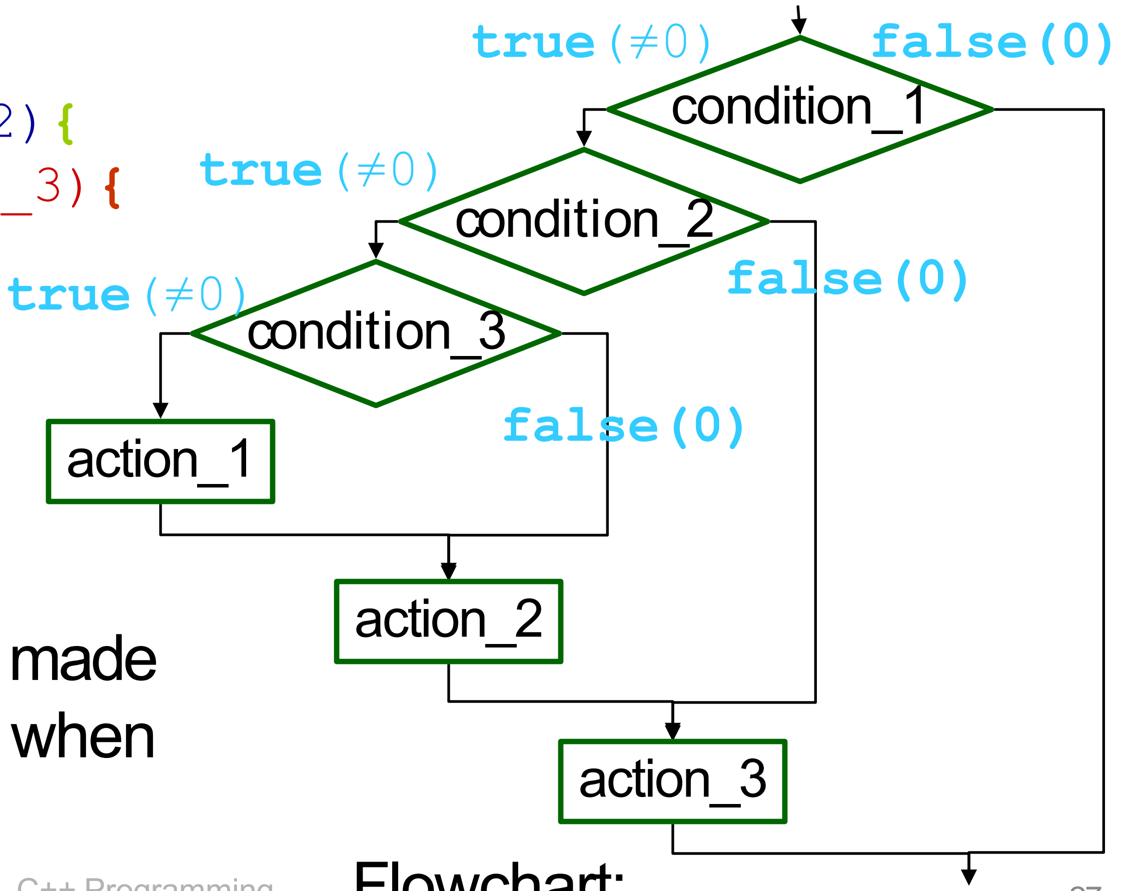


Flowchart:

Nested `if` Statements

- Nested means that one compound statement is inside another

```
if (condition_1){
     if (condition_2){
       if (condition_3){
            action_1
       }
       action_2
     }
     action_3
}
```



What changes should be made if action_4 is performed when condition_1 is false?

Flowchart:

Examples

```
if (membership == 1){
   if (age < 18)
      fee = fee * 0.5;
   if (age >= 18)
      fee = fee * 0.8;
}
```

```
if (membership == 1){
   if (age < 18)
      fee = fee * 0.5;
}
if (age >= 18)
   fee = fee * 0.8;
```

Difference?

```
if (membership == 1){
   if (age < 18)
      fee = fee * 0.5;
   else
      fee = fee * 0.8;
}
```

```
if (membership == 1)
   if (age < 18)
      fee = fee * 0.5;
   else
      fee = fee * 0.8;
```

"Dangling Else" Problem

```
if (membership == 1){
    if (age < 18)
        fee = fee * 0.5;
    else
          fee = fee * 0.8;
}
```

=

```
if (membership == 1)
    if (age < 18)
        fee = fee * 0.5;
    else
          fee = fee * 0.8;
```

"Dangling Else" Problem

This one will produce a different result. →

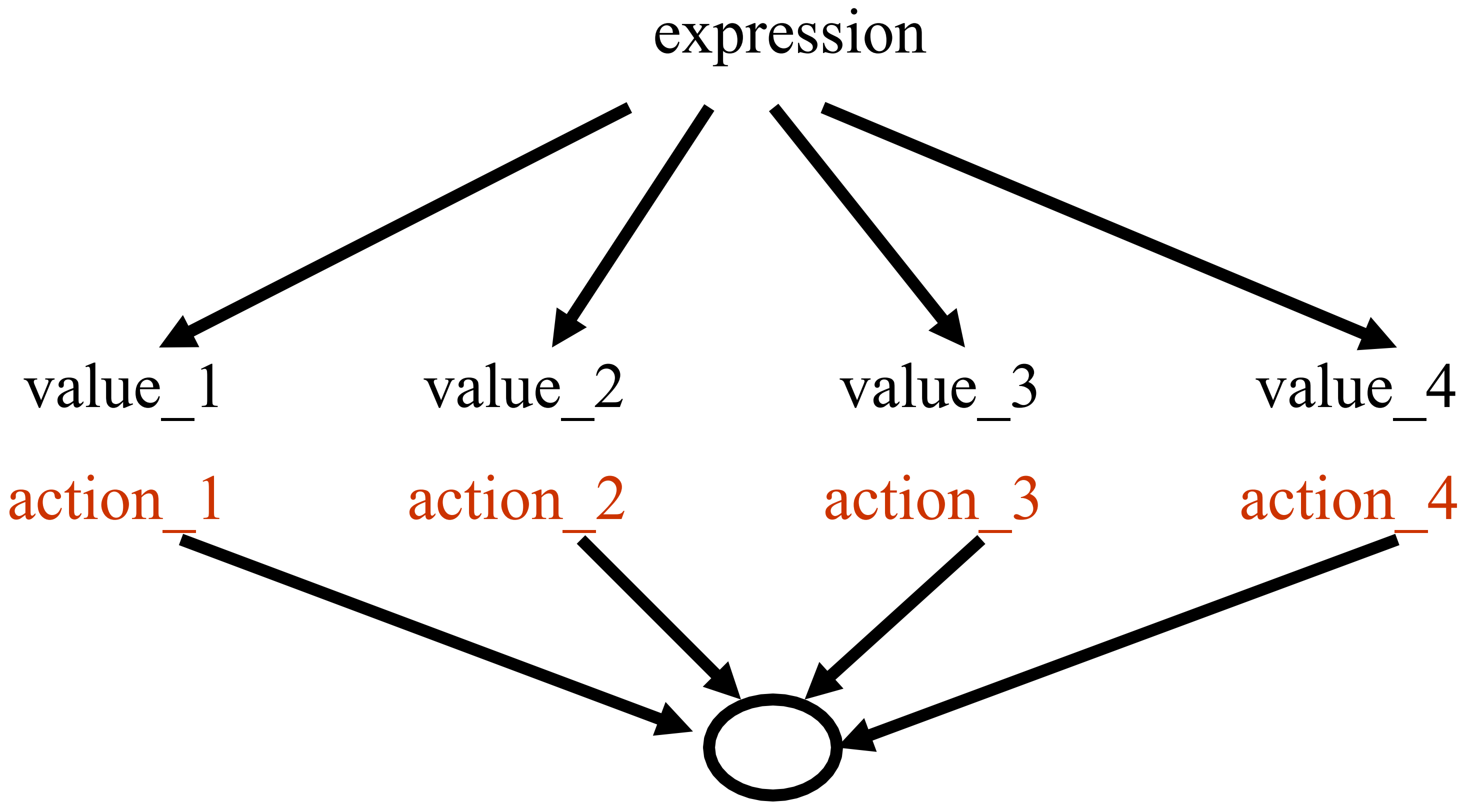
```
if (membership == 1){
    if (age < 18)
        fee = fee * 0.5;
}
else
    fee = fee * 0.8;
```

```
if (membership == 1){
    if (age < 18)
        fee = fee * 0.5;
    else
        fee = fee * 0.8;
}
```

=

```
if (membership == 1)
    if (age < 18)
        fee = fee * 0.5;
    else
        fee = fee * 0.8;
```

Multi-way Selection: `Switch` Statement



`Switch` Statement

Syntax:

```
switch (expression) {
   case value_1:   action_1;
                     break;
   case value_2:   action_2;
                     break;
   case value_3:    action_3
                     break;
   default: action_4;
  }
```

Meaning:

- Evaluate expression.
- The expression can only be a simple constant or a constant expression.
- Match case label.
- Execute sequence of statements of matching label. Until break encountered.
- Go to end of the switchstatement.
- Otherwise continue execution.

Switch Statement

- Attentions
  - The value following each **case** label must be a constant.
  - No two `case` labels can have the same value.
  - Two `case` labels may be associated with the same statements.
  - Usually include the `break` statement at the end of each case.
  - The `default` label is not required.
  - There can be only one `default` label, and it is usually put as the last.

An Example

```
char op;
int x, y;
cin >> x >> op >> y;
if(op == '+')
    cout << x << '+' << y << '=' << x + y;  else
if(op == '-')
    cout << x << '-' << y << '=' << x - y;  else if(op
== '*')
    cout << x << '*' << y << '=' << x * y;  else
if(op == '/')
    cout << x << '/' << y << '=' << x / y;  else cout
<<                  "Invalid operator!";
```

Rewrite this program using `switch` statement?

An Example

```
char op;
int x, y;

cin >> x >> op >> y;
switch (op){
   case '+':  cout << x << '+' << y << '=' << x + y;
              break;
   case '-':  cout << x << '-' << y << '=' << x - y;
              break;
   case '*':  cout << x << '*' << y << '=' << x * y;
              break;
   case '/':  cout << x << '/' << y << '=' << x / y;
              break;
   default:   cout << "Invalid operator!";
}
```

An Example

```
switch(int(score)/10){
    case 10:
    case 9: cout << "Grade = A\n";
             break;
    case 8: cout << "Grade = B\n";
            break;
    case 7: cout << "Grade = C\n";
            break;
    case 6: cout << "Grade = D\n";
            break;
    default: cout << "Grade = F\n";
}
```

- What is the output of this program if score is 95?
- What if all the "`break`" are missed?