An Assignment Statement

Used to store results of computation into a variable. Form:
variable_name = expression;

Example:

s = u*t + 0.5 * a * t * t;

Expression : can specify a formula involving constants or variables, almost as in mathematics

- If variables are specified, their values are used.
- operators must be written explicitly
- multiplication, division have higher precedence than addition, subtraction
- multiplication, division have same precedence
- addition, subtraction have same precedence
- operators of same precedence will be evaluated left to right.
- Parentheses can be used with usual meaning

Examples

```
int x=2, y=3, p=4, q=5, r, s, t;
x = r*s;                    // disaster. r, s undefined
r = x*y + p*q;
                            // r becomes 2*3 + 4*5 = 26
s = x*(y+p)*q;
                            // s becomes 2*(3+4)*5 = 70
t = x – y + p – q;
                            // equal precedence,
                            // so evaluated left to right,
                            // t becomes (((2-3)+4)-5 = -2
```

Arithmetic Between Different Types Allowed

```
int x=2, y=3, z, w;
float q=3.1, r, s;
r = x;      // representation changed
              // 2 stored as a float in r  "2.0"
z = q;      // store with truncation
              //  z takes integer value 3
s = x*q; // convert to same type,
              // then multiply
              // Which type?
```

Evaluating varA op varB
e.g. x*q

- if varA, varB have the same data type: the result will have same data type
- if varA, varB have different data types: the result will have more expressive data type
- int/short/unsigned int are less expressive than float/double
- shorter types are less expressive than longer types

Rules for storing numbers of one type into variable of another type

- C++ does the “best possible”.

```
int x; float y;
x = 2.5;
y = 123456789;
```

- x will become 2, since it can hold only integers. Fractional part is dropped.
- 123456789 cannot be precisely represented in 24 bits, so something like 1.234567 e 8 will get stored.

Integer Division

```
int x=2, y=3, p=4, q=5, u;

u = x/y + p/q;

cout << p/y;
```

- x/y : both are int. So truncation. Hence 0
- p/q : similarly 0
- p/y : 4/3 after truncation will be 1
- So the output is 1

More Examples of Division

```
int noosides=100, i_angle1, i_angle2;
i_angle1 = 360/noosides + 0.45;          // 3
i_angle2 = 360.0/noosides + 0.45;        // 4


float f_angle1, f_angle2;
f_angle1 = 360/noosides + 0.1;           // 3.1
f_angle2 = 360.0/noosides + 0.1          // 3.7
```

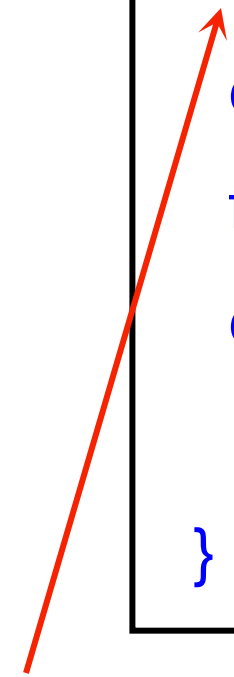
An Example Limited Precision

```
float w, y=1.5, avogadro=6.022e23;
w = y + avogadro;
```

- Actual sum : 602200000000000000000001.5
- y + avogadro will have type float, i.e. about 7 digits of precision.
- With 7 digits of precision (2^{23}), all digits after the 7^{th} will get truncated and the value of avogadro will be the same as the value of y + avogadro
- w will be equal to avogadro
- No effect of addition!

Program Example

```
main_program{
  double centigrade, fahrenheit;
  cout <<"Give temperature in Centigrade: ";
  cin >> centigrade;
  fahrenheit = centigrade * 9 / 5 + 32;
  cout << "In Fahrenheit: " << fahrenheit
       << endl;  // newline
}
```



Prompting for input is meaningless in Prutor because it is non-interactive

Re-Assignment

- Same variable can be assigned a value again
- When a variable appears in a statement, its value at the time of the execution of the statement gets used

```
int p=3, q=4, r;
r = p + q;                  // 7 stored into r
cout << r << endl;  // 7 printed as the value of r
r = p * q;                   // 12 stored into r (could be its
                              // temporary location)
cout << r << endl;  // 12 printed as the value of r
```

In C++ "=" is assignment not "equal"

```
int p=12;
p = p+1;
```

See it as: p ← p+1; // Let p *become* p+1

Rule for evaluation:

- FIRST evaluate the RHS and THEN store the result into the LHS variable
- So 1 is added to 12, the value of p
- The result, 13, is then stored in p
- Thus p finally becomes 13

p = p + 1 is nonsensical in mathematics

"=" in C++ is different from "=" in mathematics

Repeat And Reassignment

```
main_program{
    int i=1;
    repeat(10){
        cout << i << endl;
        i = i + 1;
    }
}
```

This program will print 1, 2,…, 10 on separate lines

Fundamental idiom

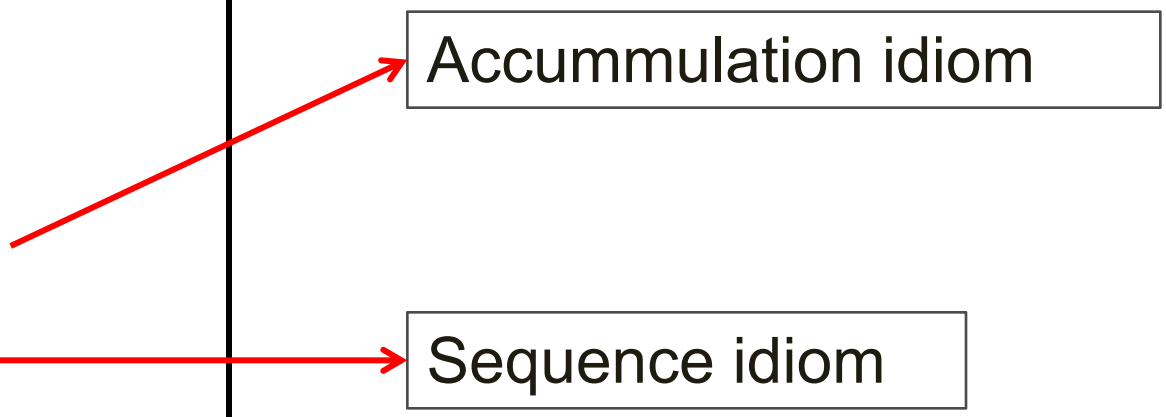
Sequence generation

- Can you make i take values 1, 3, 5, 7, …?
- Can you make i take values 1, 2, 4, 8, 16, …?
- Both can be done by making slight modifications to previous program.

Composing The Two Idioms

Write a program to calculate n! given n.

```
main_program{
  int n, nfac=1, i=1;
  cin >> n;
  repeat(n){
     nfac = nfac * i;
     i = i + 1;
  }
  cout << nfac << endl;
}
```



Finding Remainder

- x % y computes the remainder of dividing x by y
- Both x and y must be integer expressions
- Example

```
int n=12345678, d0, d1;
d0 = n % 10;                    // 8
d1 = (n / 10) % 10;             // 7
```

d0 will equal 8 (the least significant digit of n)

d1 will equal 7 (the second least significant digit of n)

Some Additional Operators

- The fragment i = i + 1 is required very frequently, and so can be abbreviated as i++

  ++ : increment operator. Unary

- Similarly we may write j-- which means j = j – 1

  -- : decrement operator. Unary

Intricacies Of ++ and --

++ and –- can be written after or before the variable. Both cause the variable to increment or decrement but with subtle differences

```
int i=5, j=5, r, s;
 r = ++i;
 s = j++;
cout << "r= " << r << " s= " << s;
```

i,j both become 6 but r is 6 and s is 5.

++ and –- can be put inside expressions but not recommended in good programming

Compound Assignment

The fragments of the form sum = sum + expression occur frequently, and hence they can be shortened to sum += expression

Likewise you may have *=, -=, …

Example

```
int x=5, y=6, z=7, w=8;

x += z;     // x becomes x+z = 12

y *= z+w; // y becomes y*(z+w) = 90
```

Blocks and Scope

- Code inside {} is called a block.
- Blocks are associated with repeats, but you may create them otherwise too.
- You may declare variables inside any block.

New summing program:

- The variable `term` is defined close to where it is used, rather than at the beginning. This makes the program more readable.
- But the execution of this code is a bit involved.

```
// The summing program
// written differently.

main_program{
   int s = 0;
   repeat(10){
         int term;
         cin >> term;
         s = s + term;
   }
   cout << s << term
<< endl;
}
```

Shadowing and scope

- Variables defined outside a block can be used inside the block, if no variable of the same name is defined inside the block.
- If a variable of the same name is defined, then from the point of definition to the end of the block, the newly defined variable gets used.
- The new variable is said to "shadow" the old variable.
- The region of the program where a variable defined in a particular definition can be used is said to be the scope of the definition.

Example

```
main_program{
   int x=5;
   cout << x << endl;   // prints 5
   {
     cout << x << endl; // prints 5
     int x = 10;
     cout << x << endl; // prints 10
   }
   cout << x << endl; // prints 5
}
```

Concluding Remarks

Variables are regions of memory which can store values

Variables have a type, as decided at the time of creation

Choose variable names to fit the purpose for which the variable is defined

The name of the variable may refer to the region of memory (if the name appears on the left hand side of an assignment), or its value (if the name appears on the right hand side of an assignment)

Further Remarks

Expressions in C++ are similar to those in mathematics, except that values may get converted from integer to real or vice versa and truncation might happen

Truncation may also happen when values get stored into a variable

Sequence generation and accumulation are very common idioms

Increment/decrement operators and compound assignment operators also are commonly used (they are not found in mathematics)

CS 101: Computer Programming and Utilization

Jan-Apr 2017

Sunita Sarawagi
(cs101@cse.iitb.ac.in)

6: Conditional Execution

Let Us Calculate Income Tax

Write a program to read income and print income tax, using following rules

- If income ≤ 1,80,000, then tax = 0
- If income is between 180,000 and 500,000 then tax= 10% of (income - 180,000)
- If income is between 500,000 and 800,000, then tax = 32,000 + 20% of (income – 500,000)
- If income > 800,000, then tax = 92,000 + 30% of (income – 800,000)

Cannot write tax calculation program using what we have learnt so far

Outline

- Basic if statement
- if-else statement
- Most general if statement form
- switch statement
- Computig Logical expressions

Basic IF Statement

Form:

if (condition) consequent

condition: boolean expression

boolean : Should evaluate to true or false

consequent: C++ statement, e.g. assignment

If condition evaluates to true, then the consequent is executed.

If condition evaluates to false, then consequent is ignored

Conditions

- Simple condition: exp1 relop exp2

  relop : relational operator: <, <=, ==, >, >=, !=

  less than, less than or equal, equal, greater than, greater than or equal, not equal
- Condition is considered true if exp1 relates to exp2 as per the specified relational operator relop

A Better Program for our Simple Problem

```
main_program {
      float income, tax;
      cin >> income;
      if (income <= 180000)
            cout << “No tax owed.” << endl;
      else
            cout << “You owe tax.” << endl;
}
// Only one condition check
// Thus more efficient than previous
```

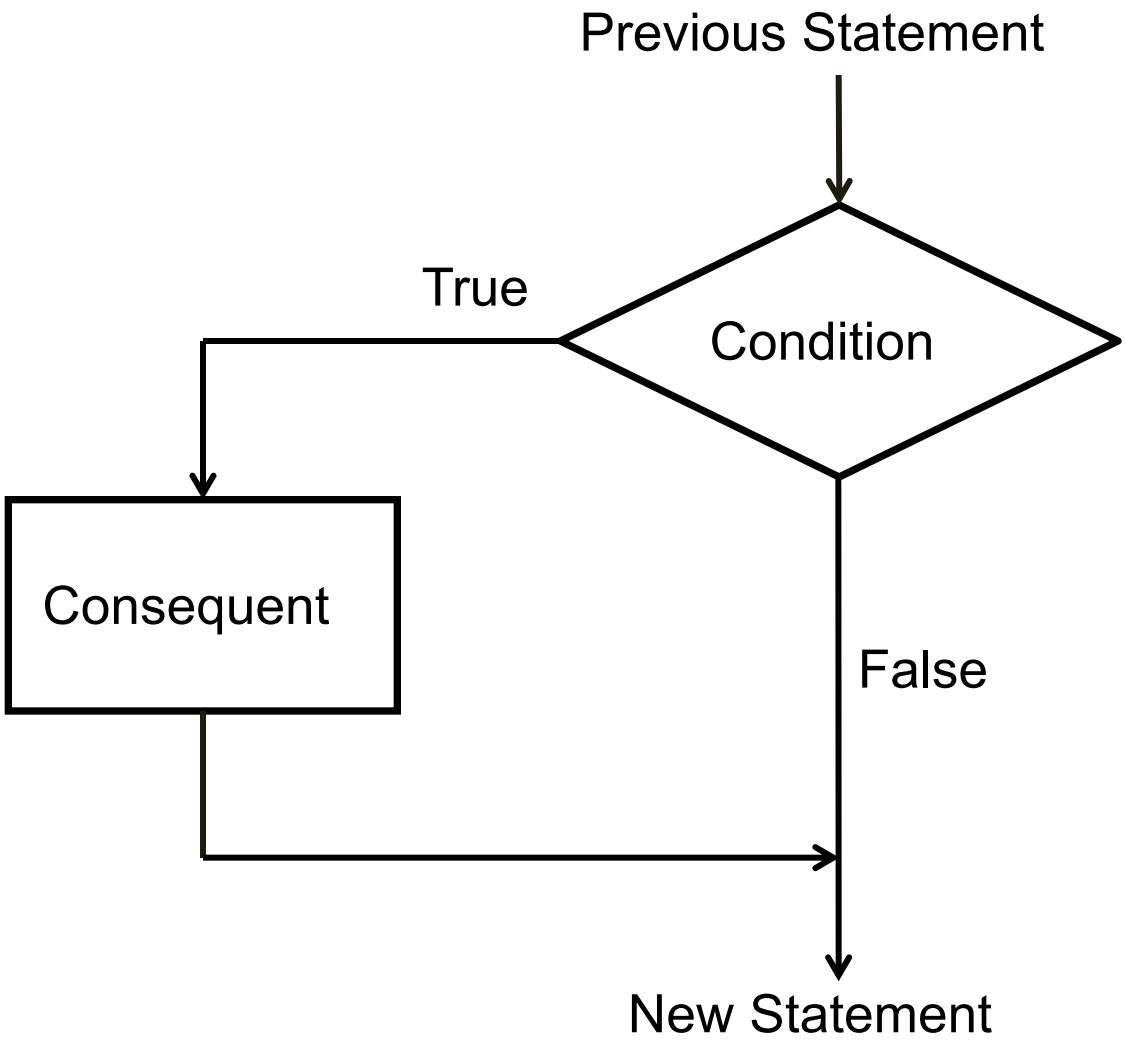
Program for the Simple Problem

```
main_program {
    float income, tax;
    cin >> income;
    if (income <= 180000)
        cout << “No tax owed” << endl;
    if (income > 180000)
    cout << “You owe tax” << endl;
}
// Always checks both conditions
// If the first condition is true,
// then you know second must be false,
// and vice versa.  Cannot be avoided
// using just the basic if statement
```

Flowchart

- Pictorial representation of a program
- Statements put inside boxes
- If box C will possibly be executed after box B, then put an arrow from B to C
- Specially convenient for showing conditional execution, because there can be more than one next statements
- Diamond shaped boxes are used for condition checks

Flowchart of the IF Statement



A More General Form of the IF Statement

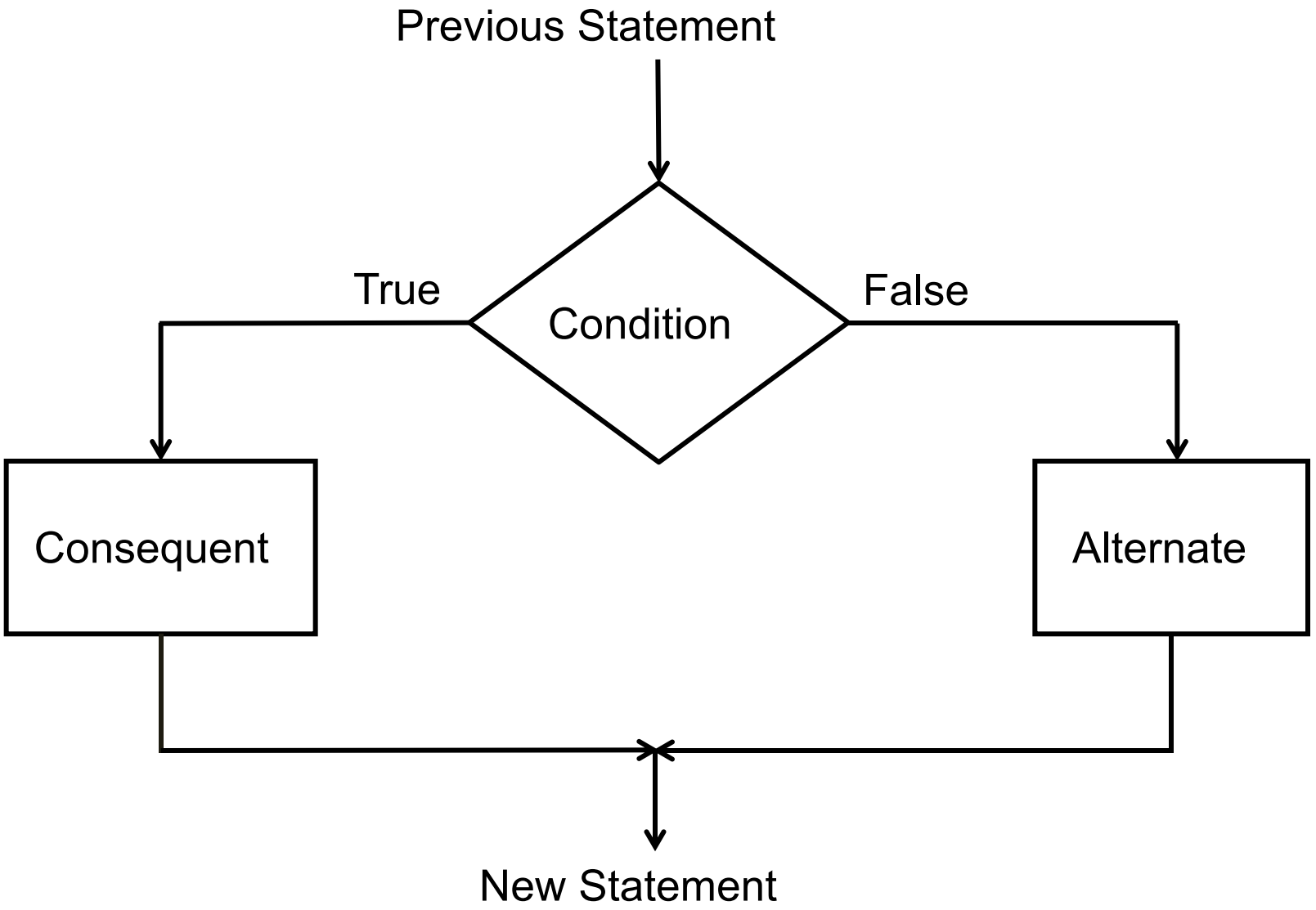
if (condition) consequent else alternate

The condition is first evaluated

If it is true, then consequent is executed

If the condition is false, then alternate is executed

Flowchart of the IF-ELSE statement



Most General Form of the IF-ELSE Statement

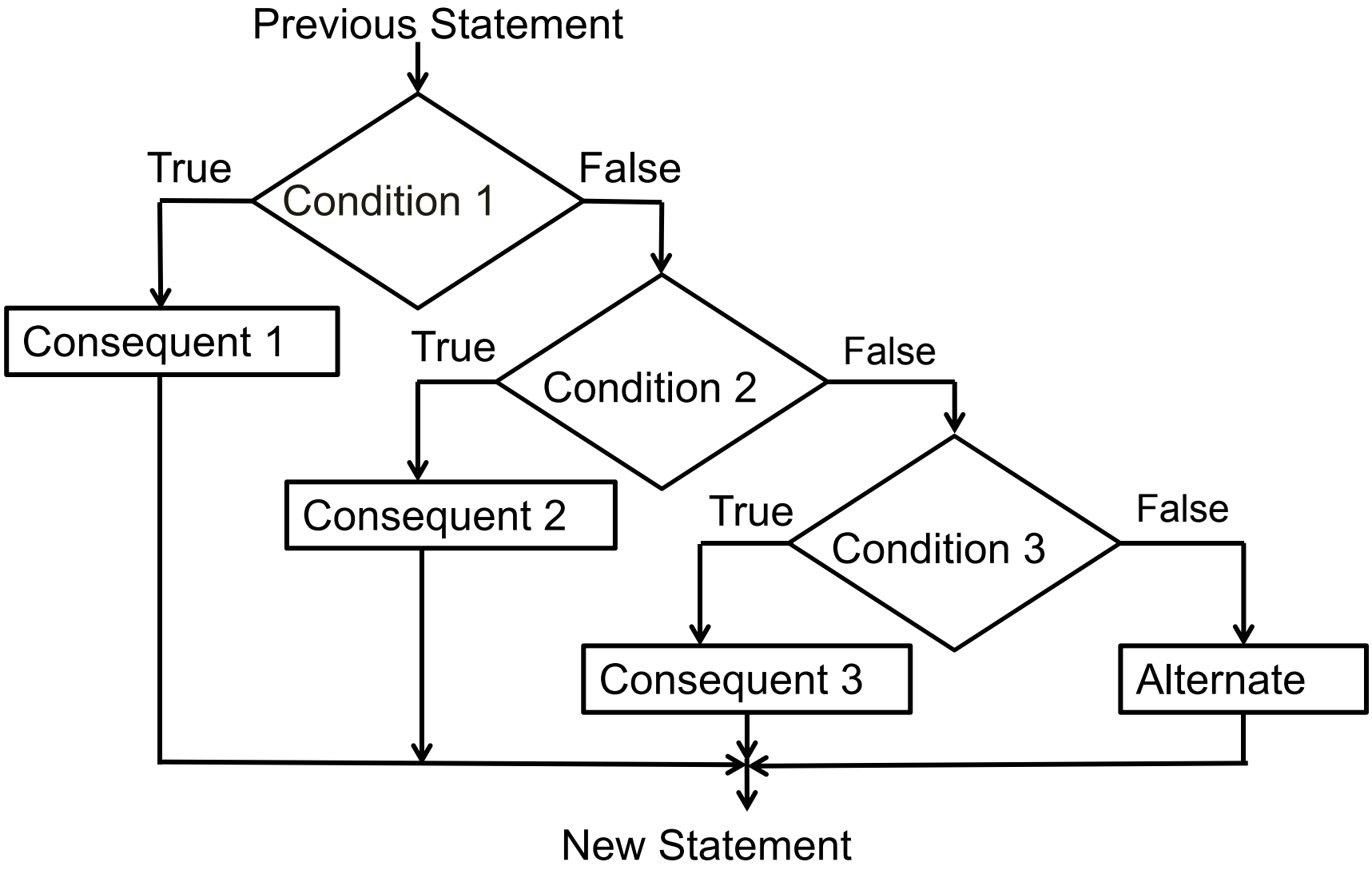
```
if (condition_1) consequent_1
else if (condition_2) consequent_2
…
else if (condition_n) consequent_n
else alternate
```

Evaluate conditions in order

Some condition true: execute the corresponding consequent. Do not evaluate subsequent conditions

All conditions false: execute alternate

Flowchart of the General IF-ELSE Statement (with 3 conditions)



Tax Calculation Program

```
main_program {
    float tax,income;
    cin >> income;
    if (income <= 180000) tax = 0;
    else if (income <= 500000)
        tax = (income – 180000) * 0.1;
    else if (income <= 800000)
        tax = (income – 500000) * 0.2 + 32000;
    else tax = (income – 800000) * 0.3 + 92000;
    cout << tax << endl;
}
```

Tax Calculation Flowchart