

CS 101: Computer Programming and Utilization

Jan-Apr 2017

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

6: Conditional Execution

About These Slides

- Based on Chapter 6 of the book *An Introduction to Programming Through C++* by Abhiram Ranade (Tata McGraw Hill, 2014)
- Original slides by Abhiram Ranade
 - First update by Varsha Apte
 - Second update by Uday Khedker

Let Us Calculate Income Tax

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

- If $\text{income} \leq 1,80,000$, then $\text{tax} = 0$
- If income is between 180,000 and 500,000 then $\text{tax} = 10\%$ of $(\text{income} - 180,000)$
- If income is between 500,000 and 800,000, then $\text{tax} = 32,000 + 20\%$ of $(\text{income} - 500,000)$
- If $\text{income} > 800,000$, then $\text{tax} = 92,000 + 30\%$ of $(\text{income} - 800,000)$

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

An Even Simpler Problem

- Using the rules given earlier, read in the income of an individual and print a message indicating whether or not the individual owes tax
- Even this simpler problem cannot be done using what we have learned so far
- For completeness, we need
 - **Sequence** of statements
default **textual appearance**
 - **Repetition** of statements
repeat statement
 - **Selection** of statements
new statement needed: **if** statement

Outline

- Basic **if** statement
- **if-else** statement
- Most general **if** statement form
- **switch** statement
- Computing 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
```

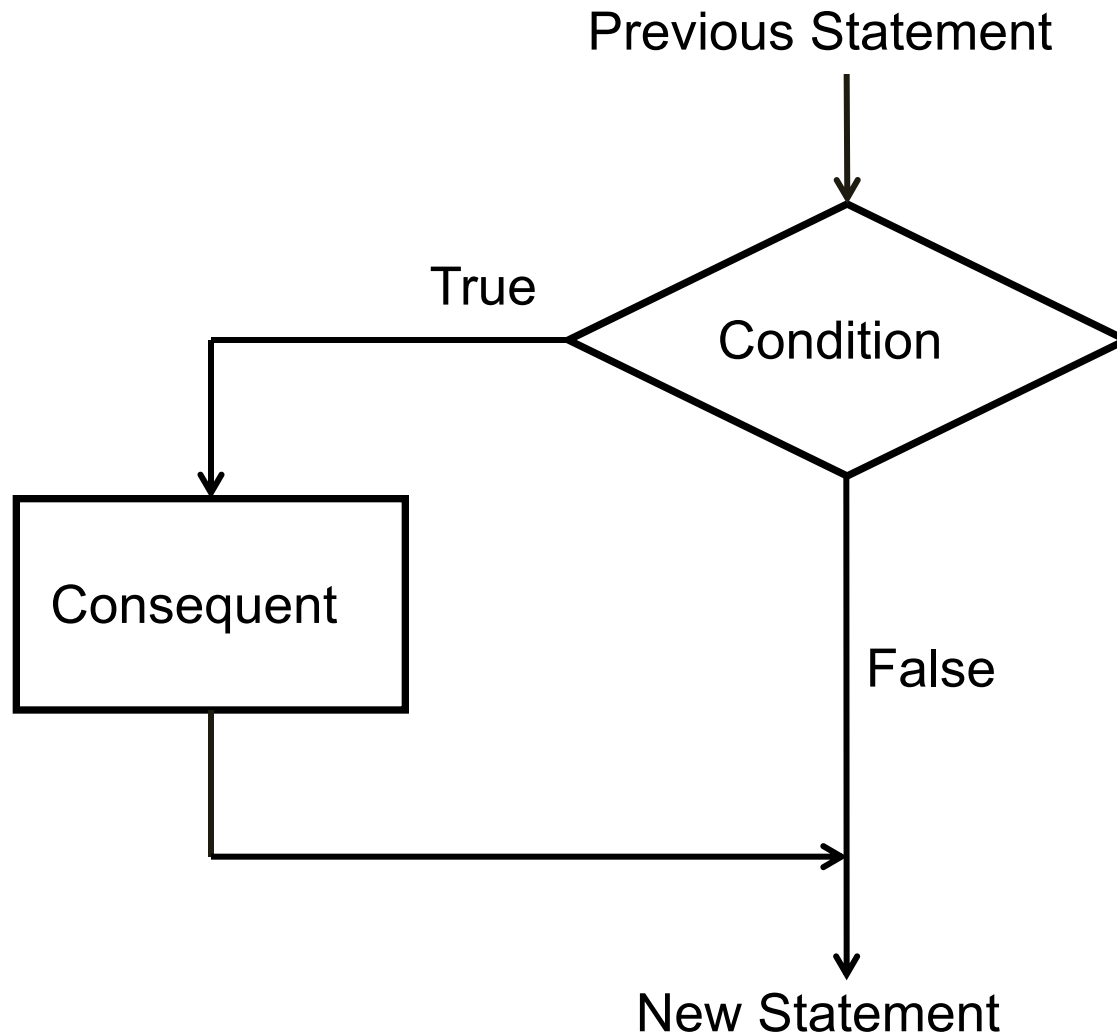

Program for the Simple Problem

```
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

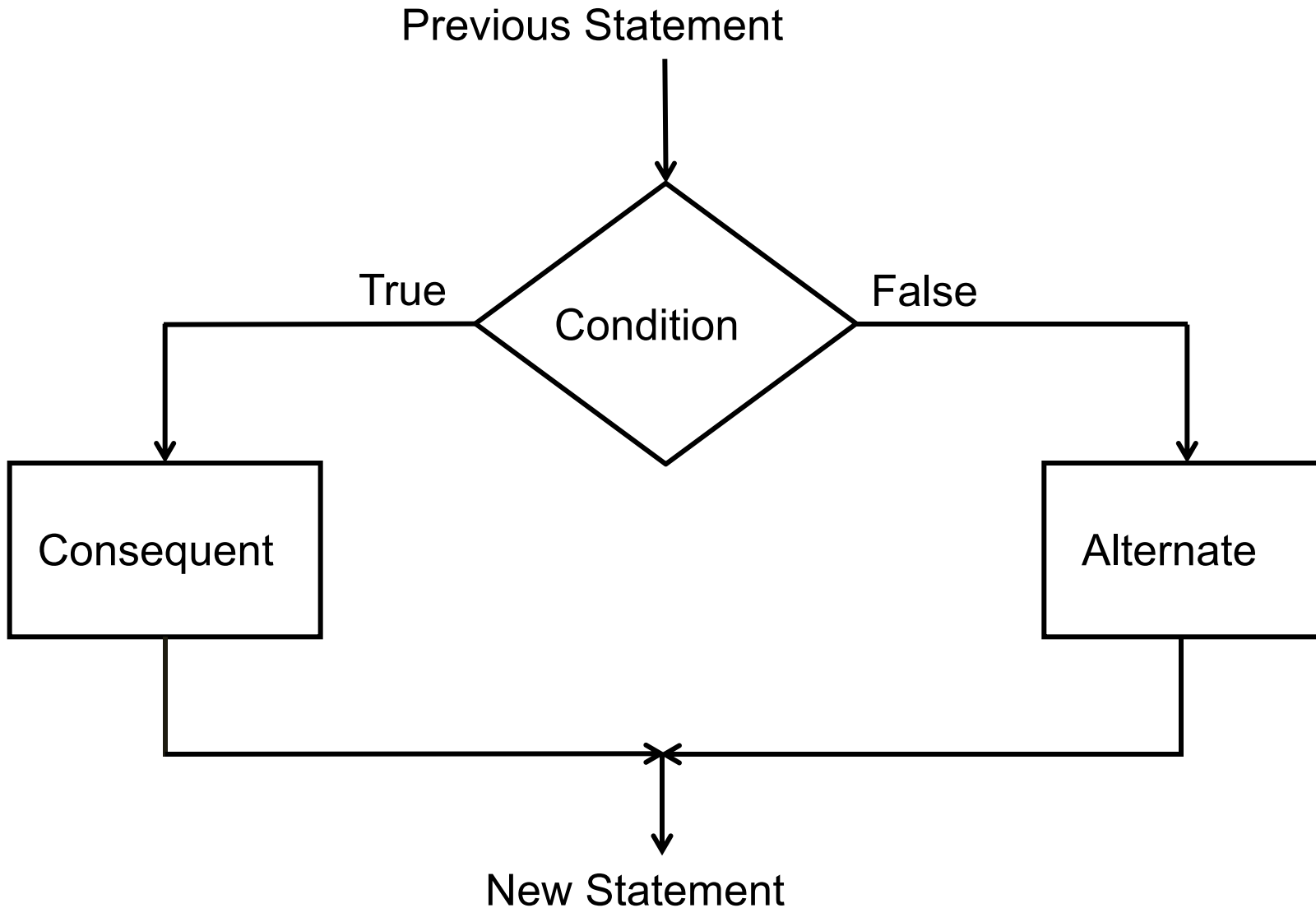
if (condition) consequent else alternate

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

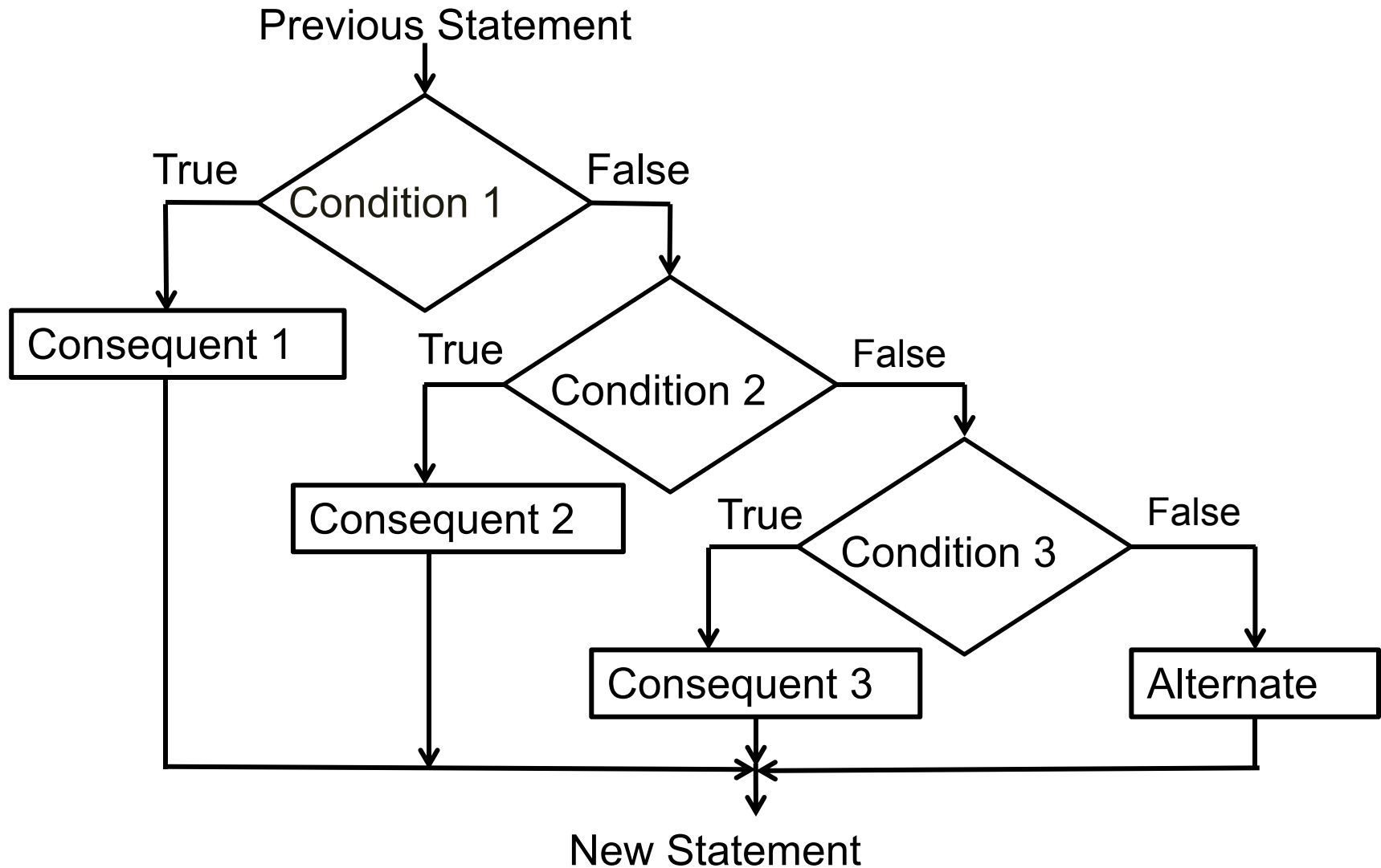
```
if (condition_1) consequent_1  
else if (condition_2) consequent_2  
...  
else if (condition_n) consequent_n  
else alternate
```

Evaluate conditions in order

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

All conditions false: execute alternate

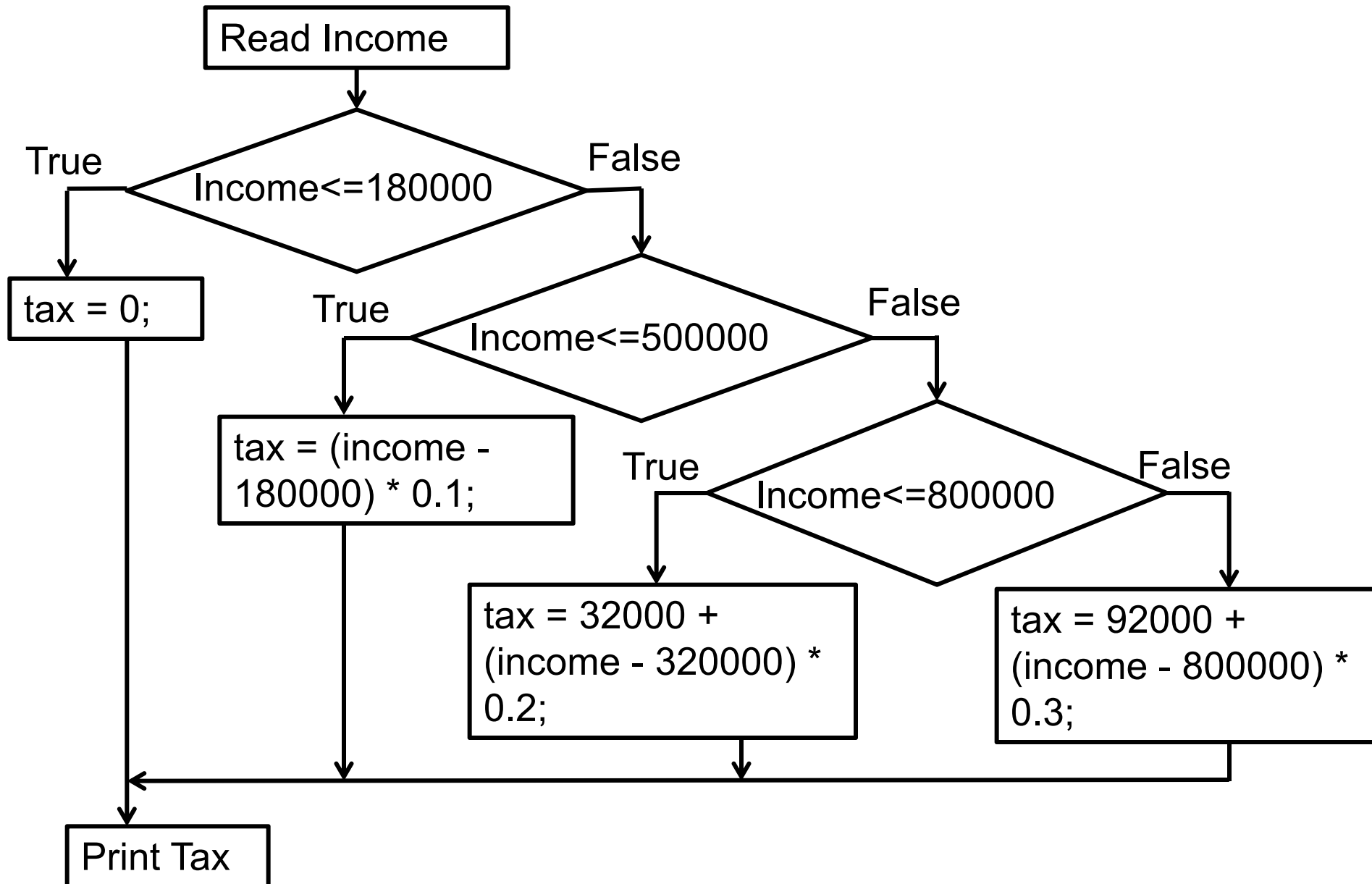
Flowchart of the General IF-ELSE Statement (with 3 conditions)



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



More General Conditions

- `condition1 && condition2` : true only if both true

Boolean AND

- `condition1 || condition2` : true only if at least one is true

Boolean OR

- `! condition` : true if only if condition is false
- Components of general conditions may themselves be general conditions, e.g.

`!((income < 18000) || (income > 500000))`

- Exercise: write tax calculation program using general conditions wherever needed

Remark

The consequent in an **if** statement can be a **block** containing several statements. If the condition is true, all statements in the block are executed, in order

Likewise the alternate

Example: If income is greater than 800000, then both the statements below get executed

```
if (income > 800000){  
    tax = 92000 + (income - 800000)*0.3;  
    cout << "In highest tax bracket.\n";  
}
```

\n : Newline character. Another way besides **endl**

Logical Data

- We have seen that we can **evaluate** conditions, combine conditions
- Why not allow storing the results (**true** or **false**) of such computations?
- Indeed, C++ has data type **bool** into which values of conditions can be stored
- The type **bool** is named after George Boole, who formalized the manipulation of logical data
- An **int** variable can have 2^{32} values, a **bool** variable can have only two values (**true/false**)

The Data Type Bool

```
bool highincome, lowincome;
```

Declares variables highincome and lowincome of type bool

```
highincome = (income > 800000);
```

```
bool fun = true;
```

Will set highincome to true if the variable income contains value larger than 800000

boolean variables which have a value can be used wherever **conditions** are expected, e.g.

```
if (highincome)
```

```
    tax = ...
```

Example: Determining If a Number is Prime

- Program should take as input a number x (an integer > 1)
- Output **Number is prime** if it is, or **number is not prime** if it is not
- Steps:
 - For all numbers 2 to $x-1$, check whether any one of these is a factor of n
 - These are $x-2$ checks
 - If none, then number is prime

Example...Prime

Let's try using the accumulation idiom with a boolean variable

Be careful of = vs ==

Example...Prime

```
main_program {  
    int x; cin >> x; // read x 4534534536  
    int i = 2;        //first factor to check;  
    bool factorFound = false; // no factor found yet;  
    repeat (x-2) {  
        factorFound = factorFound || ((x % i) == 0 );  
        // Remainder is 0 when x is divisible by i  
        i++;  
    }  
    if (factorFound) cout << x << " is not prime"  
        << endl;  
}
```


Remarks

- Conditional execution makes life interesting
- Master the 3 forms of if
- Exercise: write the tax calculation program without using the general if and without evaluating conditions unnecessarily. Hint: use blocks
- You can nest if statements inside each other: some pitfalls in this are discussed in the book

SAFE quiz

- What is printed by this code snippet: `"int x=3,y=1; {int x=4; {x = x+2;} y=x;} cout << (x+y);"`
- What does this code print? `"int i=0,s=0; repeat(3) {if (i%2==0) s += i; else s += 2*i; i++;} cout << s;"`
- What does this program print? `"unsigned int x,c=0; cin>>x; repeat (32) {if (x%2==1) c++; x = x/2;} cout << c;"`
- What does this program print? `"unsigned int x,c=0; cin>>x; repeat (32) {if (x%2==1) c++; x = x/2;} cout << c;"`