CS 101:
Computer Programming and Utilizati on

Jul-Nov 2016

Bernard Menezes
(cs101@cse.iitb.ac.in)

Lecture 5: 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 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

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
  - 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
- 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

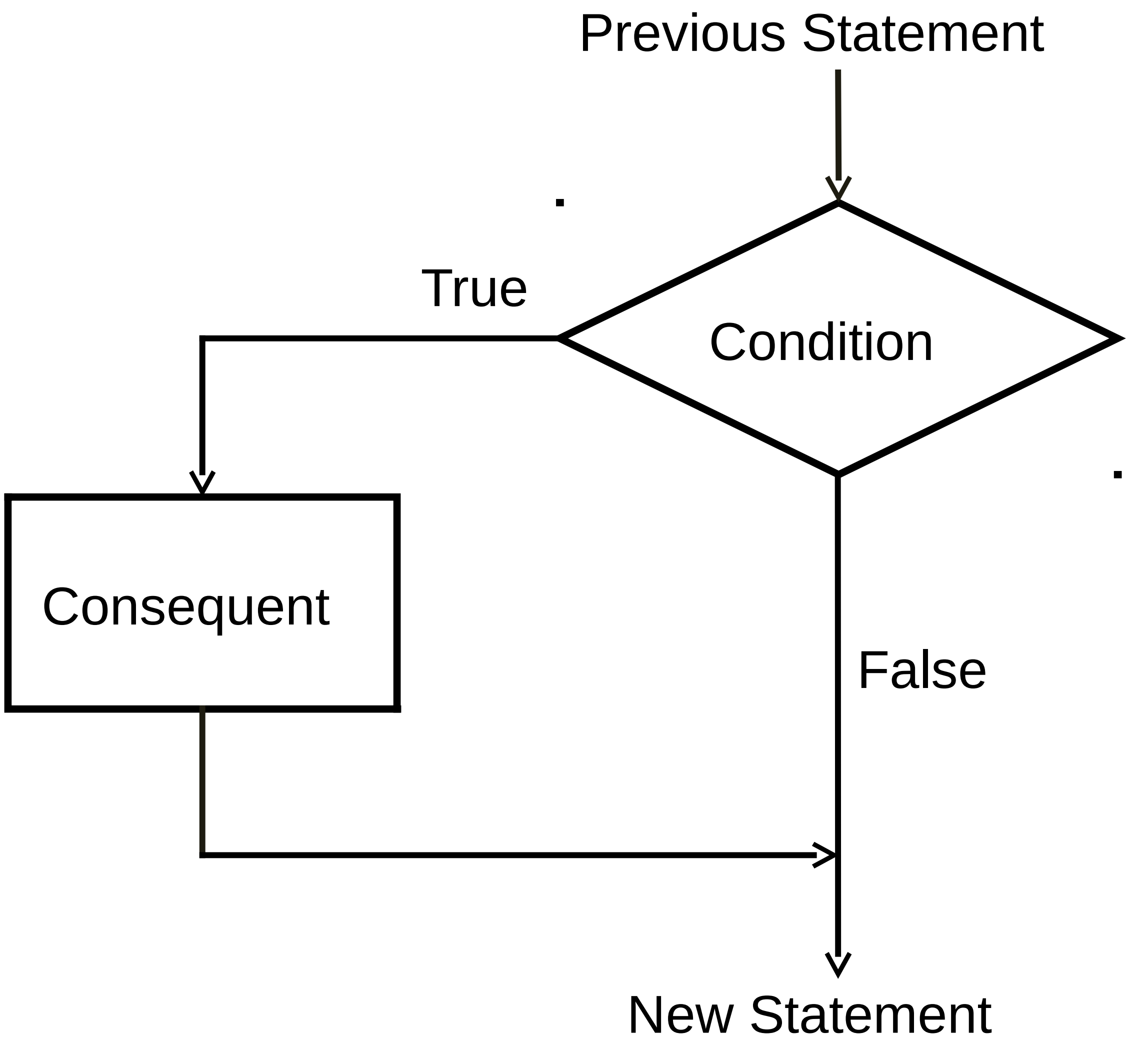
Program for the Simple Problem

```
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 (in this case)
,
// 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 Stateme nt

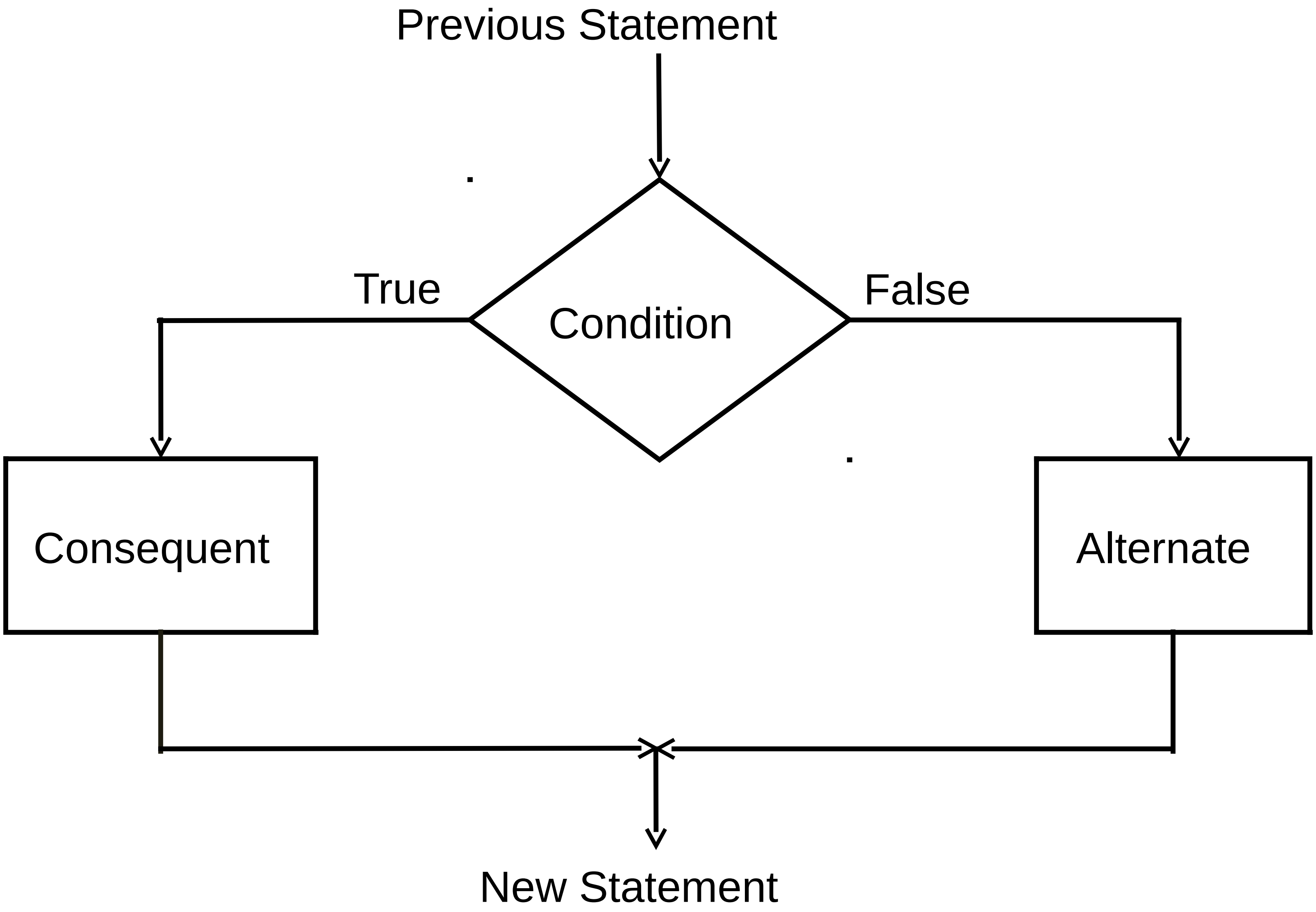
if (condition) consequent else alternate

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



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
```

Most General Form of the IF-ELSE Statement

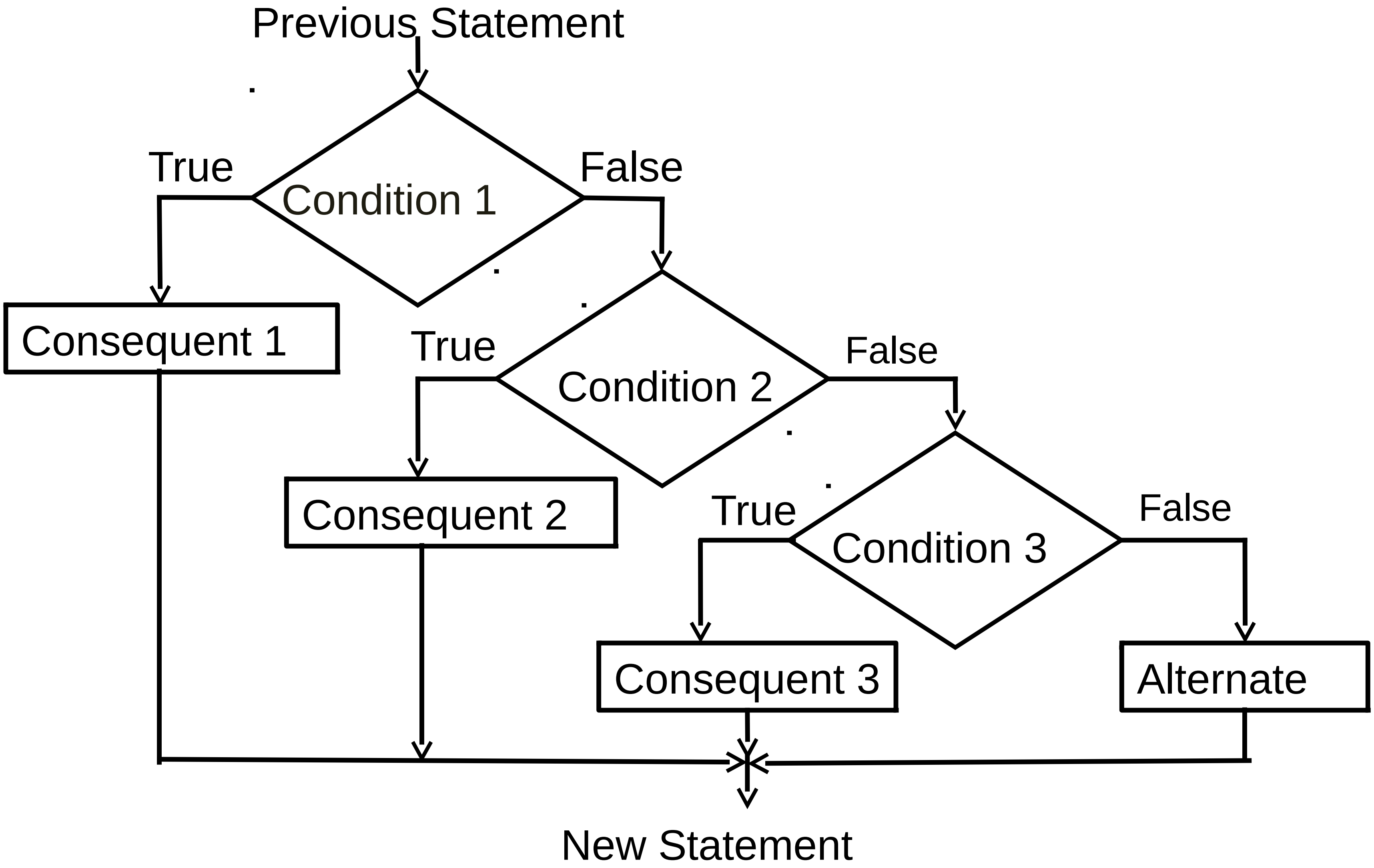
```
if (condition_1) consequent_1
else if (condition_2) consequent_2
…
else if (condition_n) consequent_n
else alternate
```

Evaluate conditions in order

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

All conditions false: execute alternate

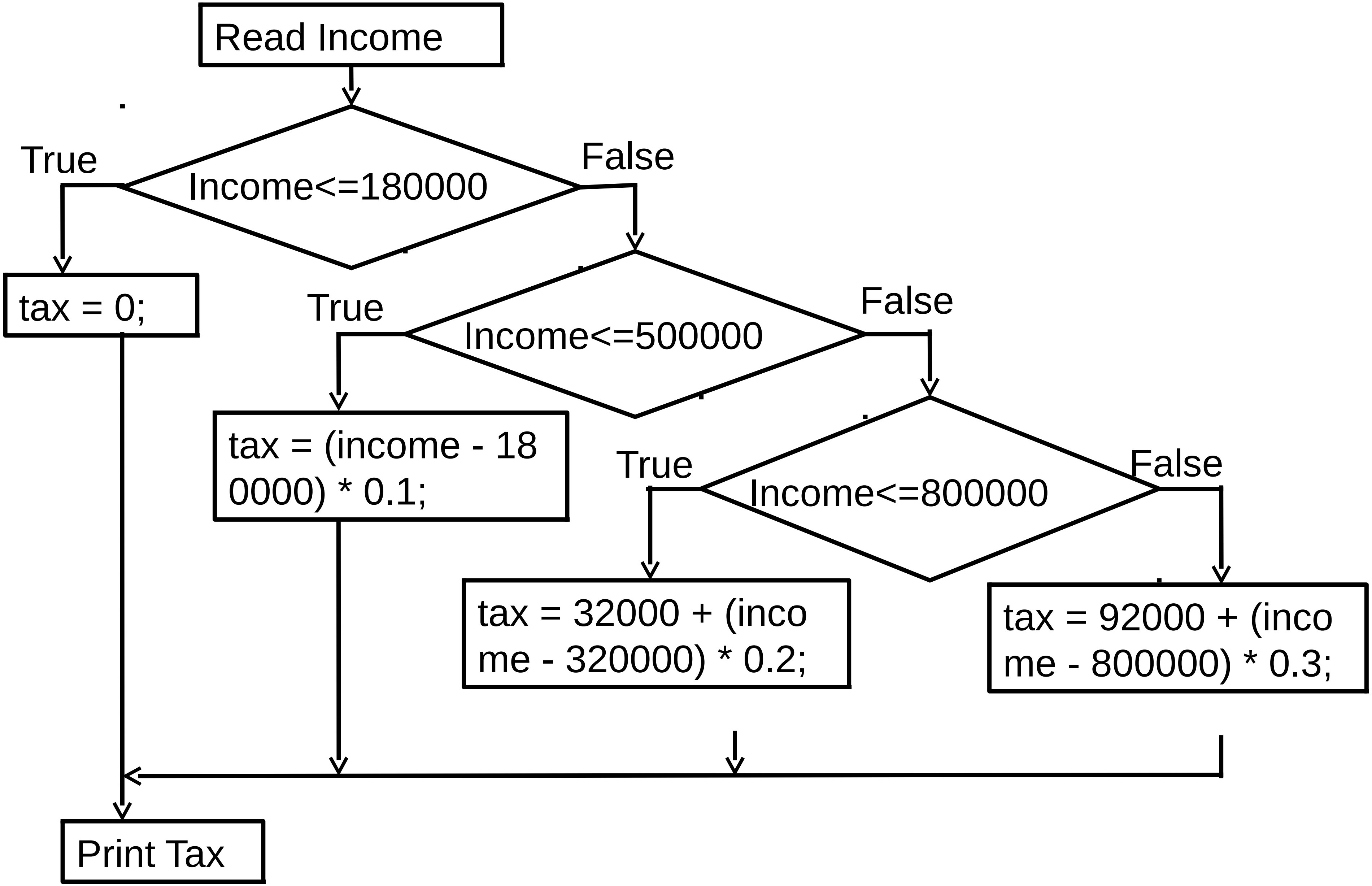
Flowchart of the General IF-ELSE Statement (with 3 conditions)



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


Read Income
Income<=180000
True
False
tax = 0;
Income<=500000
True
False
tax = (income - 180000) * 0.1;
Income<=800000
True
False
tax = 32000 + (income - 320000) * 0.2;
tax = 92000 + (income - 800000) * 0.3;
Print Tax

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 s everal 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

Blocks and Scope

- Code inside {} is called a block
- Blocks are associated with repeats, but you may create them arbitrarily
- 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 << endl;
}
```

How Definitions In A Block Execute

Basic rules

- A variable is created every time control reaches the declaration
- All variables created in a block are destroyed every time control reaches the end of the block
- Creating a variable is only notional; the compiler simply starts using that region of memory from then on
- Likewise destroying a variable is notional

Shadowing And Scope

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

Another Example of Block

```
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
}
```

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 h ave 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