The for statement: motivation

- Example: Write a program to print a table of cubes of numbers from 1 to 100.

```
int i = 1;
repeat(100){
      cout << i <<‘ ‘<< i*i*i << endl;
      i++;
}
```

- This idiom: do something for every number between x and y occurs very commonly.
- The *for* statement makes it easy to express this idiom, as follows:

```
for(int i=1; i<= 100; i++)
  cout << i <<‘ ‘<< i*i*i << endl;
```

- We will see how this works next.

The for statement

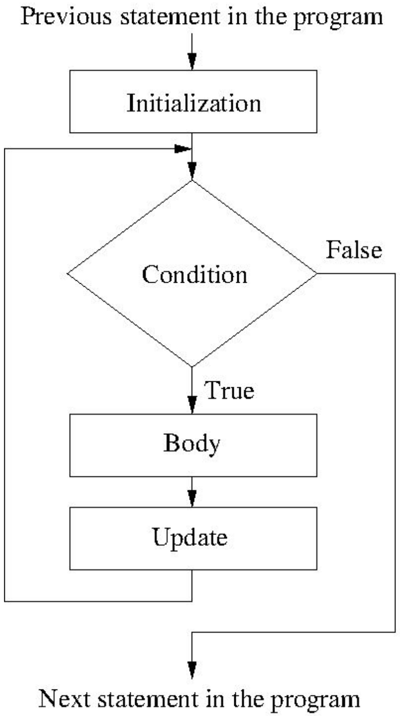
Form: ***for(initialization; condition; update) body***

- ***initialization, update*** : Typically assignments (no semi-colon).
- ***condition*** : boolean expression.

Execution:

- Before the first iteration of the loop the ***initialization*** is executed.
- Within each iteration:
  - ***condition*** is first tested.
  - If it fails, the loop execution ends.
  - If the ***condition*** succeeds, then the ***body*** is executed.
  - After that the ***update*** is executed. Then the next iteration begins.
- Flowchart given next.

Flowchart for for


Previous statement in the program
Initialization
Condition
False
True
Body
Update
Next statement in the program

```
for(initialization;
    condition;
    update)
  body

for(int i=1;
    i <= 100;
    i++)
  cout <<i<<‘ ‘<<i*i*i<<endl;
```

Remarks

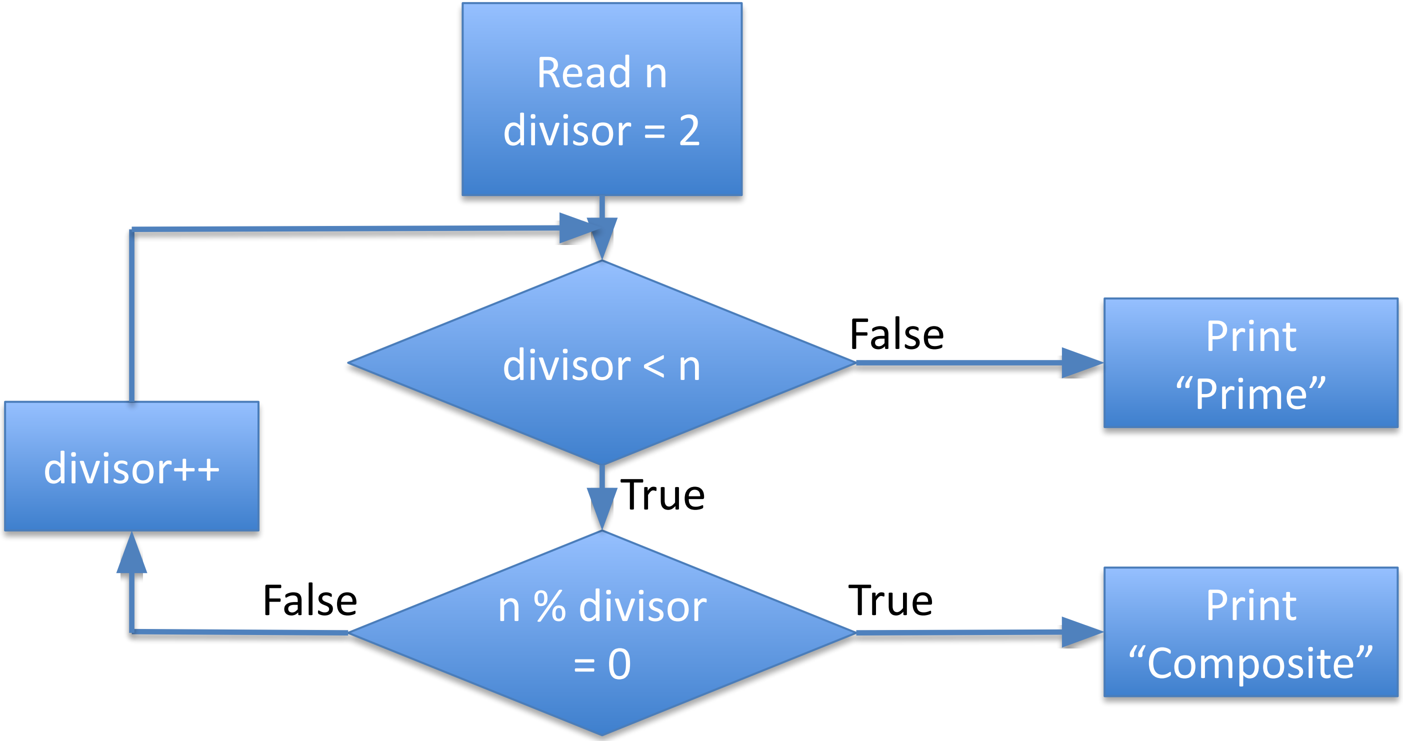
- New variables can be defined in ***initialization***. These variables are accessible inside the loop body, including ***condition*** and ***update***, but not outside.
- Variables defined outside can be used inside, unless shadowed by new variables.
- ***Break*** and ***continue*** can be used, with natural interpretation.
- Typical use of ***for***: a single variable is initialized and updated, and the condition tests whether it has reached a certain value. Such a variable is called the control variable of the ***for*** statement.

Determining whether a number n is prime

Simple manual algorithm: Check whether any of the numbers between 2 and n-1 divides n.

Will improve upon what we did last week.

- Make a flowchart of the manual algorithm.
- See if it can be put into the format of the ***for*** statement.



Read n
divisor = 2

divisor < n

False

Print
“Prime”

True

divisor++

False

n % divisor
= 0

True

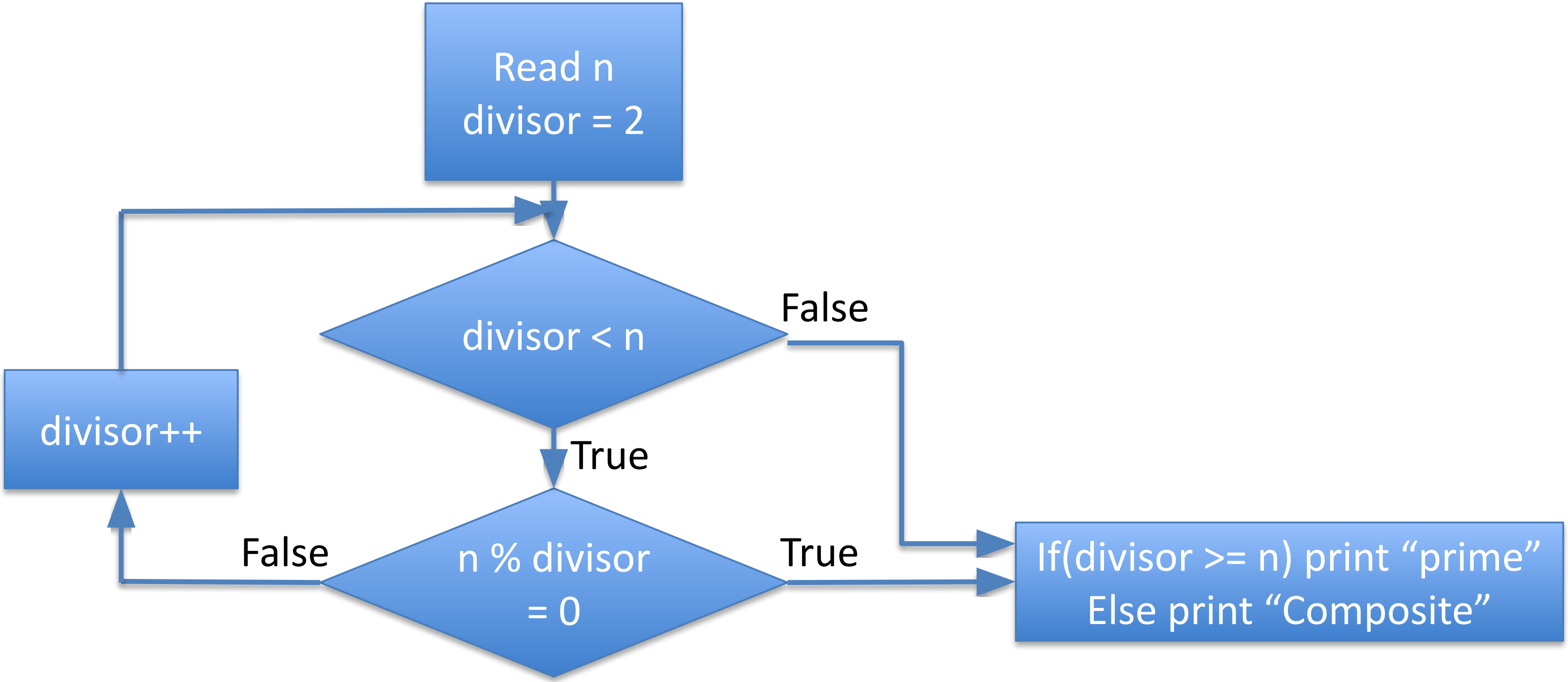
Print
“Composite”

Remarks

- The previous flowchart is functionally correct and faithfully represents what you do manually.

However, flowcharts are expected to be “structured”

- Should not have paths flowing all over the page.
- Typically, the flow should be:
  - Sequence of steps
  - Some of the steps can be loops, which may contain loops..
  - Single start point, single end point
- Our flowchart has two paths going out, i.e. 2 end points.
  - We should try to avoid this.



Read n
divisor = 2

divisor < n

False

True

divisor++

n % divisor
= 0

False

True

If(divisor >= n) print “prime”
Else print “Composite”

Program to test primality

```
main_program{
  int n; cin >> n;
  int divisor = 2;
  for( ; divisor < n; divisor++){
    if(n % divisor == 0) break;
  }
  if(divisor >= n) cout <<“Prime”<<endl;
  else cout <<“Composite”<<endl;
}
```

Remarks

- We have left the “initialization” part of the for statement empty – this is allowed.
- We could have placed divisor = 2 in the initialization.
- However, we could not have placed “int divisor = 2” in the initialization – then the variable divisor would not be available outside the loop, in the last statement.

Exercise: What will this program print?

```
main_program{
  int n; cin >> n;
  int divisor = 2;
  for(int divisor=2 ; divisor < n; divisor++){
    if(n % divisor == 0) break;
  }
  if(divisor >= n) cout <<“Prime”<<endl;
  else cout <<“Composite”<<endl;
}
```

Exercise

- Write a program that prints out the sequence 1, 2, 4, ... 65536.
  - Hint: The update part of the for does not have to be addition, it can be other operations too.

What we discussed

- Often we need to iterate such that in each iteration a certain variable takes a simple sequence of values, i.e. variable i goes form 1 to n.
- In such a case the for statement is very useful
- The variable whose values form the sequence is called a “control variable” for the loop.
- Matching the flow chart of the manual algorithm to the structure of the while or for takes some work.