Introduction to Programming

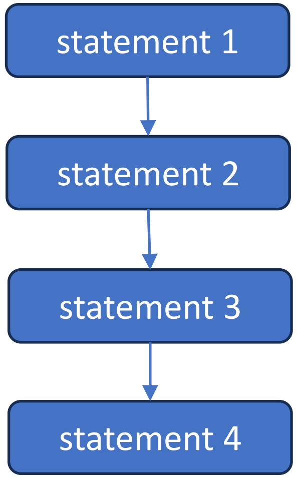
Chapter 3

Conditionals and loops

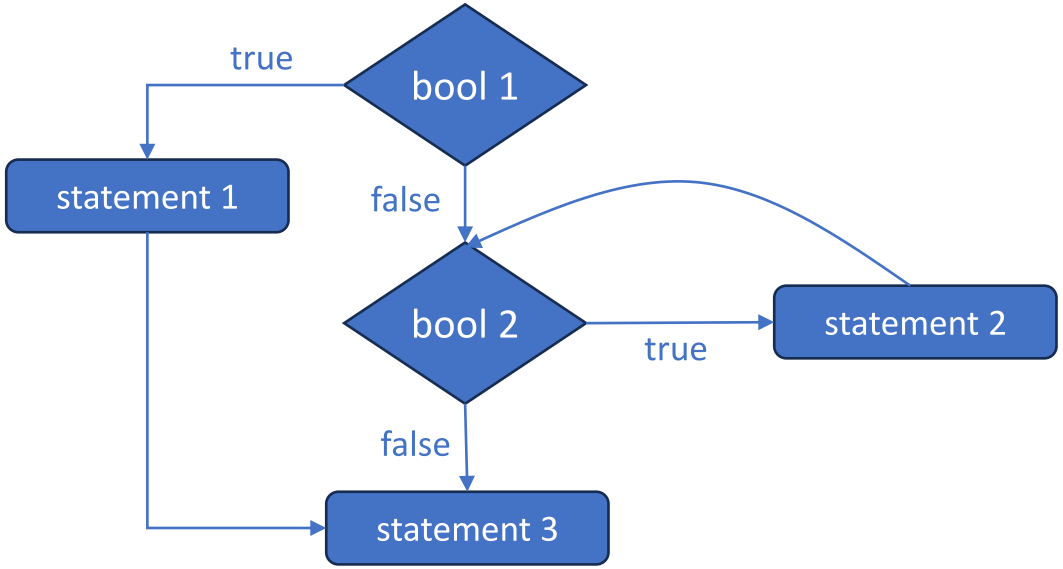
Conditionals and loops

Control flow

- The sequence of statements that are actually executed in a program.
- Conditionals and loops enable us to choreograph control flow.



linear control flow
[previous lectures]



conditionals & loops
[this lectures]

Conditionals

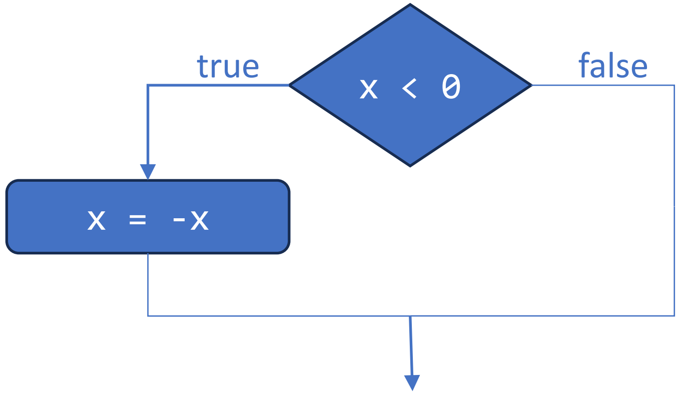
The `if` statement

The `if` statement

Execute certain statements depending on the values of certain variables.

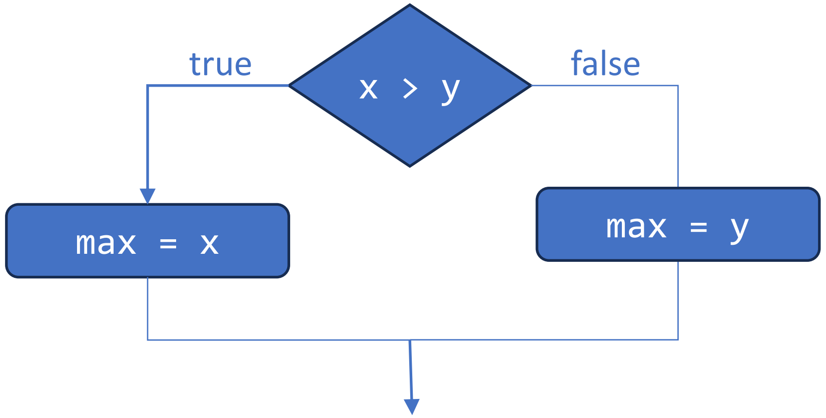
- Evaluate a boolean expression.
- If `true`, execute a statement.
- The `else` option: If `false`, execute a different statement.

Example:
```
if (x < 0) x = -x;
```



Replaces x with absolute value of x

Example:
```
if (x > y) max = x;
else       max = y;
```



Computes the maximum of x and y

Example of `if` statement use: simulate a coin flip

```
#include<iostream>
#include<cstdlib>

int main() {
    srand(time(0));

    if(rand()%2 == 0)
        std::cout << "Heads\n";
    else
        std::cout << "Tails\n";
}
```

Example of `if` statement use: 2-sort

Q. What does this program do?

```
#include<iostream>

int main() {
    int a, b;
    std::cin >> a >> b;

    if(a > b) {
        int t = a;
        a = b;
        b = t;
    }

    std::cout << a << " " << b <<std::endl;
}
```

A. Reads two integers from the command line, then prints them out in numerical order.

Pop quiz on `if` statement

Q. Add code to this program that puts a, b, and c in numerical order.

```
#include<iostream>

int main() {
    int a, b, c;
    std::cin >> a >> b >> c;






    std::cout << a << " " << b << " " << c << "\n";
}
```

```
➢ a
  123 99 1
  1 99 123

➢ a
  99 1 123
  1 99 123
```

Pop quiz on `if` statement

Q. Add code to this program that puts a, b, and c in numerical order.

A.

```
#include<iostream>

int main() {
    int a, b, c;
    std::cin >> a >> b >> c;

    if (b < a)
    { int t = a; a = b; b = t; }
    if (c < a)
    { int t = a; a = c; c = t; }
    if (c < b)
    { int t = b; b = c; c = t; }

    std::cout << a << " " << b << " " << c << "\n";
}
```

makes a smaller than b

makes a smaller than both b and c

makes b smaller than c

```
➢ a
  123 99 1
  1 99 123

➢ a
  99 1 123
  1 99 123
```

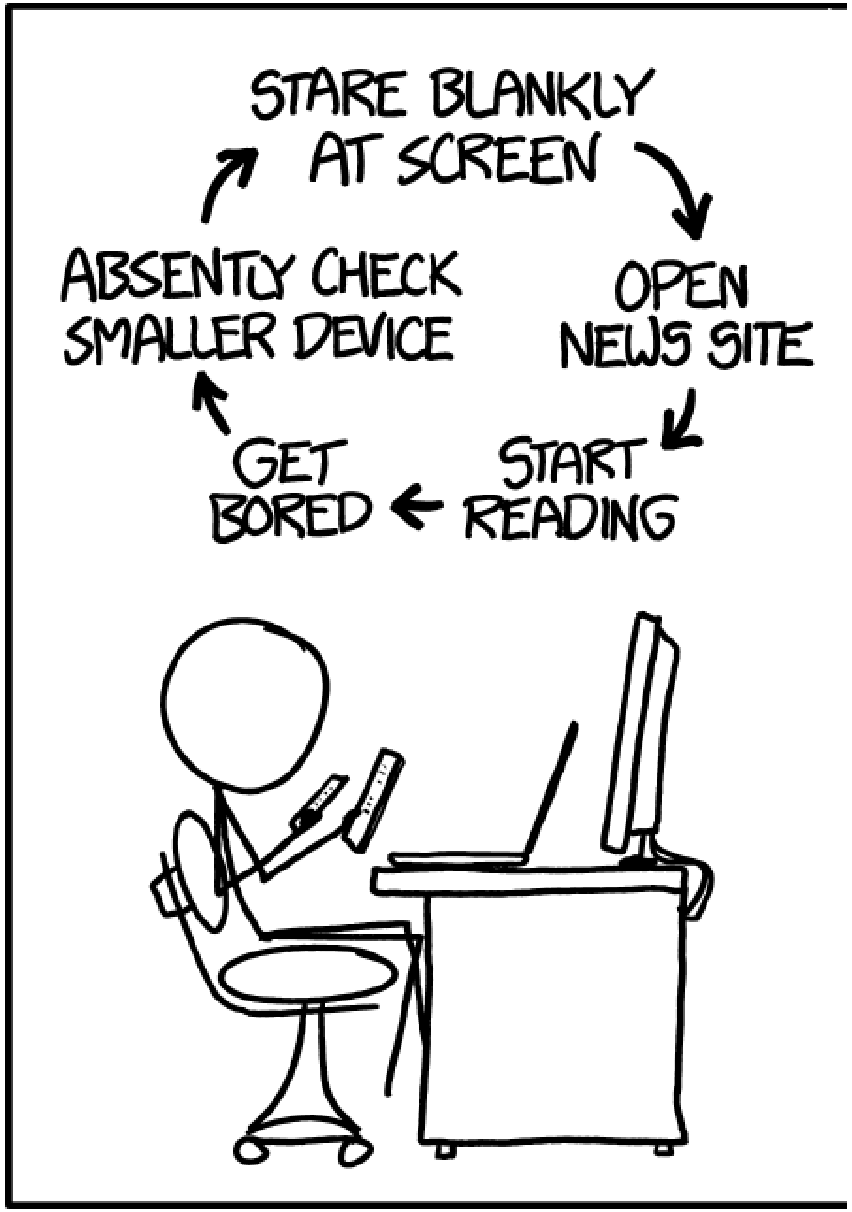
Example of `if` statement use: error checks

```
#include<iostream>

int main() {
    int a, b;
    std::cin >> a >> b;

    std::cout << a << " + " << b << " = " << a + b << "\n";
    std::cout << a << " * " << b << " = " << a * b << "\n";
    if(b == 0)
        std::cout << "Division by zero\n";
    else {
        std::cout << a << " / " << b << " = " << a / b << "\n";
        std::cout << a << " % " << b << " = " << a % b << "\n";
    }
}
```

Good programming practice. Use conditionals to check for and avoid runtime errors.

Loops

The `while` and `for` statements



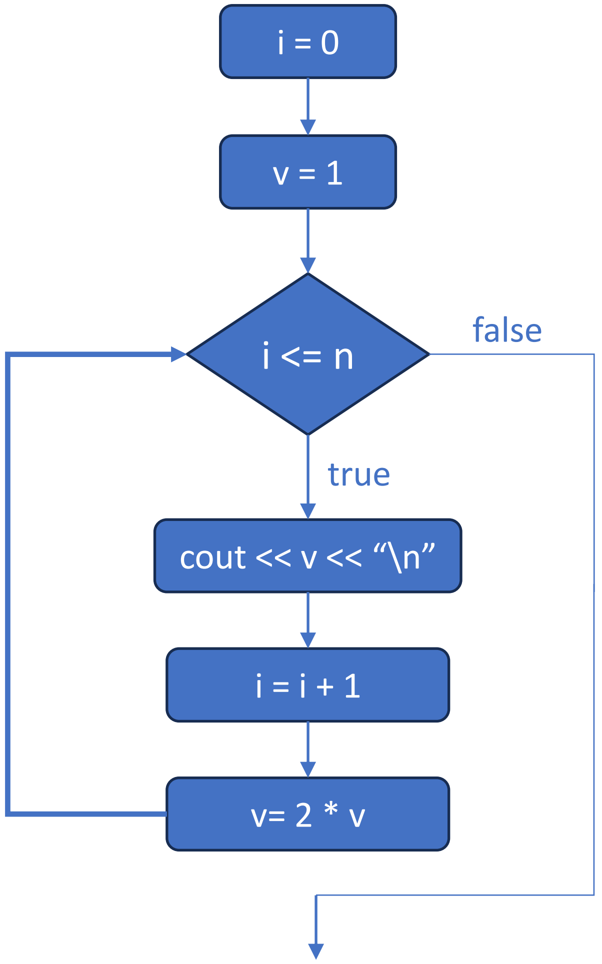
The `while` loop

Execute certain statements repeatedly until certain conditions are met

- Evaluate a boolean expression.
- If `true`, execute a sequence of statements.
- Repeat.

```
int i = 0;
int v = 1;
while (i <= n) {
   std::cout << v << "\n";
   i = i + 1;
   v = 2 * v;
}
```

Prints the powers of two from 2^0 to 2^n



Example of **`while`** loop use: print powers of two

```
#include<iostream>

int main() {
    int n;
    std::cin >> n;
    int i = 0;
    int v = 1;
    while (i <= n) {
       std::cout << v << "\n";
       i = i + 1;
       v = 2 * v;
    }
}
```

Prints the powers of two from 2^0 to 2^n

A trace is a table of variable values after each statement.

`i`	`v`	`i <= n`
0	1	true
1	2	true
2	4	true
3	8	true
4	16	true
5	32	true
6	64	true
7	128	false

values at the beginning of each iteration [assuming n is 6]

```
➢ a
6
1
2
4
8
16
32
64
```

output shown in blue

Pop quiz on `while` loops

Q. Anything wrong with the following code?

```
#include<iostream>

int main() {
    int n;
    std::cin >> n;

    int i = 0;
    int v = 1;
    while (i <= n)
       std::cout << v << "\n";
       i = i + 1;
       v = 2 * v;

}
```

Pop quiz on `while` loops

Q. Anything wrong with the following code?

```
#include<iostream>

int main() {
    int n;
    std::cin >> n;

    int i = 0;
    int v = 1;
    while (i <= n) {
       std::cout << v << "\n";
       i = i + 1;
       v = 2 * v;
    }
}
```

A. Yes! Needs braces.

Q. What does it do (without the braces)?

A. Goes into infinite loop!

```
➢ a
6
1
1
1
1
1
1
1
1
1
1
1
1
```

challenge: figure out how to stop it on your computer

Example of while loop use: implement `sqrt()`

Goal. Implement square root function.

Newton-Raphson method to compute $\sqrt{c}$

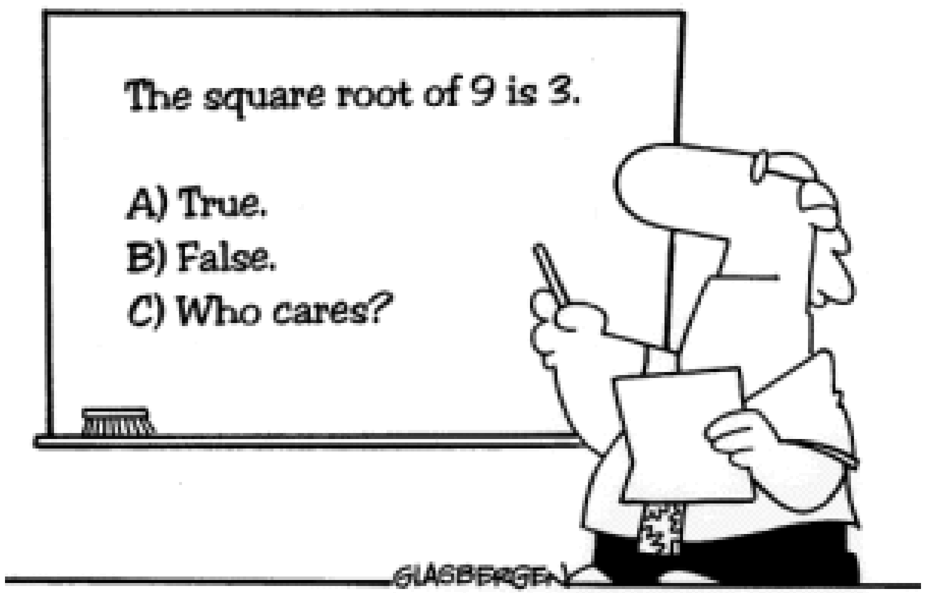
- Initialize $t_0 = c$
- Repeat until $t_i = \frac{c}{t_i}$ (up to desired precision):

  Set t_{i+1} to be the average of t_i and $\frac{c}{t_i}$

i	t_i	$\frac{2}{t_i}$	*average*
0	2	1	1.5
1	1.5	1.3333333	1.4166667
2	1.4166667	1.4117647	1.4142157
3	1.4142157	1.4142114	1.4142136
4	1.4142136	1.4142136	

computing the square root of 2 to seven places

Copyright 1996 Randy Glasbergen. www.glasbergen.com



Many students actually look forward to Mr. Atwadder's math tests.

Example of while loop use: implement `sqrt()`

Newton-Raphson method to compute $\sqrt{c}$

- Initialize $t_0 = c$
- Repeat until $t_i = \frac{c}{t_i}$ (up to desired precision):

  Set t_{i+1} to be the average of t_i and $\frac{c}{t_i}$

```cpp
#include<iostream>
#include<cmath>
#include<iomanip>

int main() {
    double c; std::cin >> c;
    double EPS = 1E-15;
    double t = c;
    while (std::abs(t - c/t) > t*EPS)
            t = (c/t + t) / 2.0;
    std::cout << std::setprecision(16) << t;
}
```

Scientists studied computation well before the onset of the computer

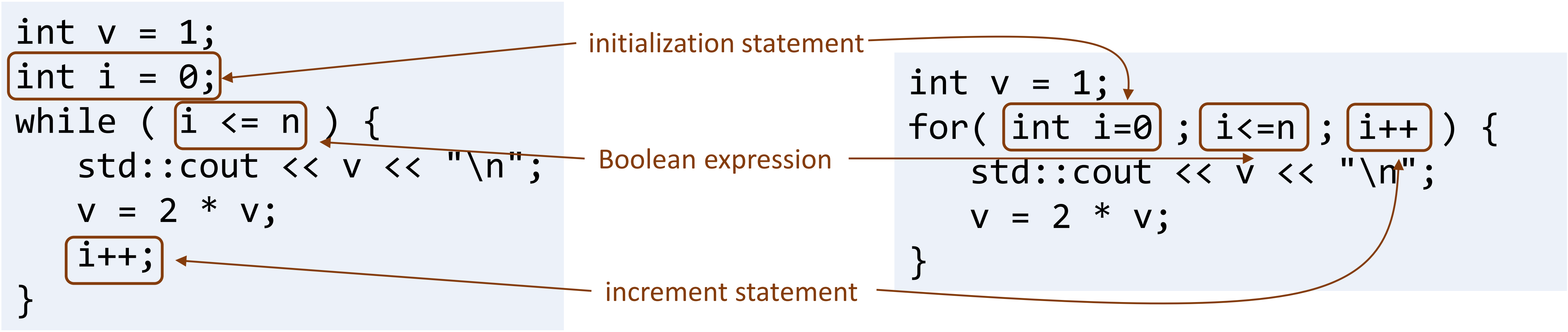
Isaac Newton
1642—1727

```
➢ a
60481729
7777

➢ a
2
1.414213562373095
```

An alternative: the `for` loop

An alternative repetition structure. ← Why? Can provide code that is more compact and understandable.

- Evaluate an initialization statement.
- Evaluate a Boolean expression.
- If `true`, execute a sequence of statements, then execute an increment statement.
- Repeat.



```
int v = 1;
int i = 0;
while ( i <= n ) {
    std::cout << v << "\n";
    v = 2 * v;
    i++;
}
```

initialization statement

Boolean expression

increment statement

```
int v = 1;
for( int i=0 ; i<=n ; i++ ) {
    std::cout << v << "\n";
    v = 2 * v;
}
```

Prints the powers of two from 2^0 to 2^n

Examples of **for** loop use

```
int sum = 0;
for (int i = 1; i <= N; i++)
   sum += i;
std::cout << sum << "\n";
```

Compute sum $(1 + 2 + 3 + \cdots + N)$

i	sum
1	1
2	3
3	6
4	10

trace at end of loop
for N = 4

```
long long product = 1;
for (int i = 1; i <= N; i++)
   product *= i;
std::cout << product << "\n";
```

Compute $N! = 1 * 2 * 3 * \cdots * N$

i	product
1	1
2	2
3	6
4	24

k	$\frac{2\pi k}{N}$
0	0
1	1.5708
2	3.14159
3	4.71239
4	6.28319

```
for (int k = 0; k <= N; k++)
   std::cout << k << " " << 2*std::numbers::pi*k/N << "\n";
```

Print a table of function values

```
int v = 1;
while (v <= N/2)
    v = 2*v;
std::cout << v << "\n";
```

v
2
4
8
16

trace
N=18

Print largest power of 2 less than or equal to N

Example of `for` loop use: subdivisions of a ruler

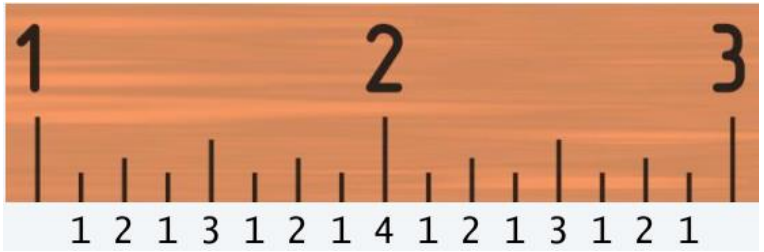
Create subdivisions of a ruler to 1/N inches.

- Initialize `ruler` to one space.
- For each value `i` from 1 to N:

  sandwich `i` between two copies of `ruler`.

```
#include<iostream>
#include<string>

int main() {
    int N; std::cin >> N;
    std::string ruler = " ";
    for (int i = 1; i <= N; i++)
        ruler = ruler + std::to_string(i) + ruler;
    std::cout << ruler;
}
```

Note: Small program can produce huge amount of output.



i	ruler
1	`" 1 "`
2	`" 1 2 1"`
3	`" 1 2 1 3 1 2 1"`
4	`" 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 "`

End-of-loop trace

```
➢ a
100
terminate called after throwing an instance
of 'std::bad_alloc'
```

2^{100} – 1 integers in output!

Pop quiz on `for` loops

Q. What does the following program print?

```
int f = 0, g = 1;
for (int i = 0; i <= 10; i++) {
   std::cout << f << "\n";
   f = f + g;
   g = f - g;
}
```

Pop quiz on `for` loops

Q. What does the following program print?

```
int f = 0, g = 1;
for (int i = 0; i <= 10; i++) {
   std::cout << f << "\n";
   f = f + g;
   g = f - g;
}
```

A.

Beginning-of-loop trace

i	f	g
0	0	1
1	1	0
2	1	1
3	2	1
4	3	2
5	5	3
6	8	5
7	13	8
8	21	13
9	34	21
10	55	34

values printed

Example: Finding binary representation of a number

Problem: Print a number in binary.

Nesting

Putting a statement inside another

Nesting conditionals and loops

Nesting

- Any “statement” within a conditional or loop may itself be a conditional or a loop statement.
- Enables complex control flows.
- Adds to challenge of debugging.

Example:

```
for (int t = 0; t < trials; t++)
{
    int cash = stake;
    while (cash > 0 && cash < goal)
        if (rand() % 2) cash++;
        else            cash--;
    if (cash == goal) wins++;
}
```

if-else statement within a while loop within a for loop

[Stay tuned for an explanation of this code.]

Example of nesting conditionals: Tax rate calculation

Goal. Given income, calculate proper tax rate.

income (pkr)	rate (%)
0 – 600,000	0
600,000 – 1,200,000	2.5
1,200,000 – 2,400,000	12.5
2,400,000 – 3,600,000	22.5
3,600,000 – 6,000,000	27.5
6,000,000 +	35.0

```
if (income <= 600’000) rate = 0.0;
else
   {
     if (income <= 1’200’000) rate = 0.025;
     else
        {
          if (income <= 2’400’000) rate = 0.125;
          else
             {
               if (income <= 3’600’000) rate = 0.225;
               else
                  {
                     if (income <= 6’000’000) rate = 0.275;
                     else                     rate = 0.35;
                  }
             }
        }
  }
```

if statement
within an if statement

if statement
within an if statement
within an if statement
within an if statement
within an if statement

Pop quiz on nested `if` statements

Q. Anything wrong with the following code?

```
#include<iostream>

int main() {
     double income; std::cin >> income;
     double rate = 0.35;
     if (income <=   600'000) rate = 0.0;
     if (income <= 1'200'000) rate = 0.025;
     if (income <= 2'400'000) rate = 0.125;
     if (income <= 3'600'000) rate = 0.225;
     if (income <= 6'000'000) rate = 0.275;
     std::cout << rate << "\n";
}
```

Pop quiz on nested `if` statements

Q. Anything wrong with the following code?

```
#include<iostream>

int main() {
     double income; std::cin >> income;
     double rate = 0.35;
     if (income <=   600'000) rate = 0.0;
else if (income <= 1'200'000) rate = 0.025;
else if (income <= 2'400'000) rate = 0.125;
else if (income <= 3'600'000) rate = 0.225;
else if (income <= 6'000'000) rate = 0.275;
     std::cout << rate << "\n";
}
```

Note. Braces are not needed in this case, but BE CAREFUL when nesting `if-else` statements because of potential ambiguity

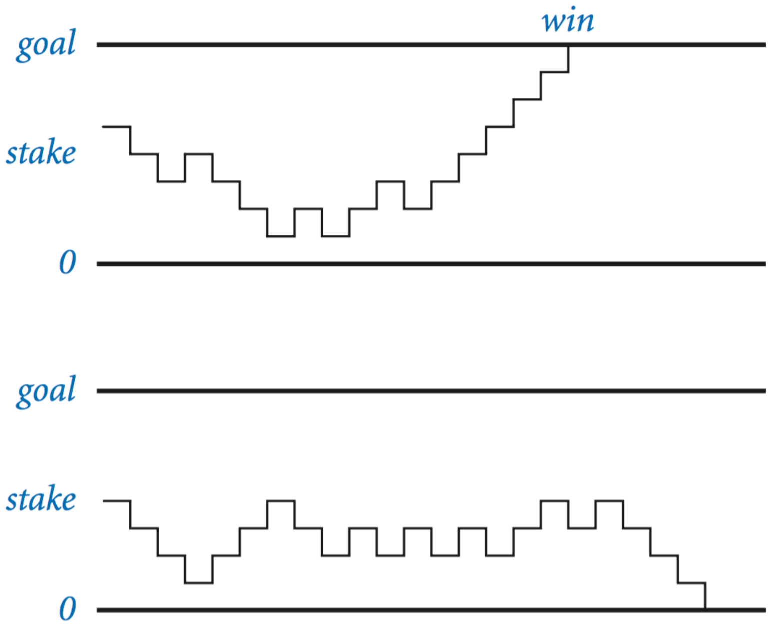
A. Yes! Need else clauses. Without them, code is equivalent to:

```
if (income < 6’000’000) rate = 0.275;
else                    rate = 0.35;
```

Gambler’s ruin problem

A gambler starts with *stake* PKR and places 1 PKR fair bets.

- Outcome 1 (loss): Gambler goes broke with 0 PKR.
- Outcome 2 (win): Gambler reaches *goal* PKR.



Gambler simulation sequences

Q. What are the chances of winning?
Q. How many bets until win or loss?

One approach: *Monte Carlo simulation*.

- Use a simulated coin flip.
- Repeat and compute statistics.

Example of nesting conditionals and loops: Simulate gambler's ruin

Gambler's ruin simulation

Seed pseudo-random generator and read input.

Run all the experiments.

Run one experiment.

Make one bet.

If goal met, count the win.

Print #wins and # trials.

```
#include<iostream>
#include<cstdlib>

int main() {
    srand(time(0));
    int stake, goal, trials;
    std::cin >> stake >> goal >> trials;

    int wins = 0;
    for (int t = 0; t < trials; t++) {
        int cash = stake;
        while (cash > 0 && cash < goal) {
                if (rand() % 2) cash++;
                else            cash--;
        }
        if (cash == goal) wins++;
    }
    std::cout << wins << " wins of " << trials;
}
```

for loop

while loop within a for loop

if statement within a while loop within a for loop

Digression: simulation and analysis

Facts (known via mathematical analysis for centuries)

- Probability of winning = stake ÷ goal.
- Expected number of bets = stake × desired gain.

Example

- 20% chance of turning $500 into $2500.
- Expect to make 1 million $1 bets

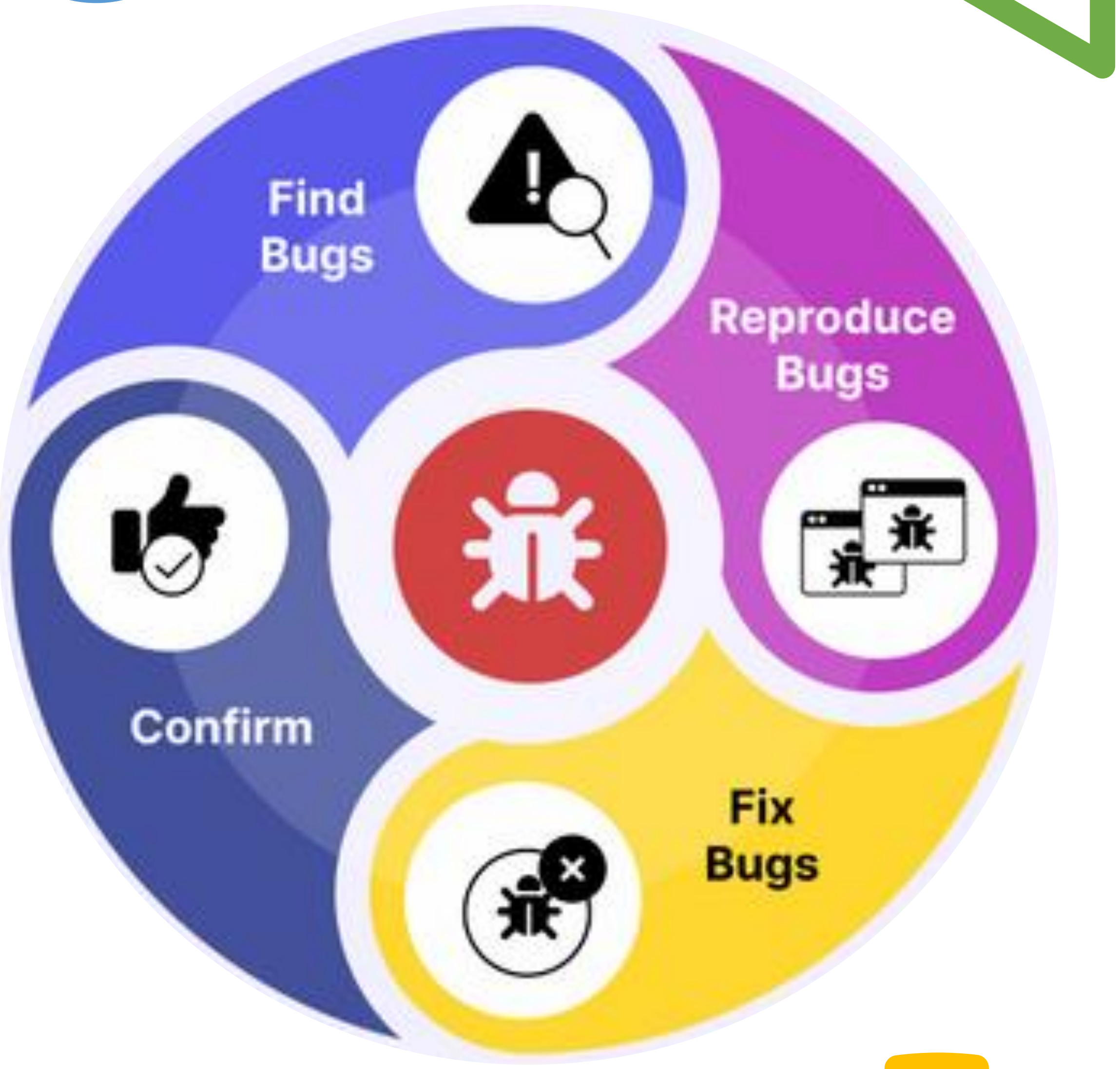
$$500/2500 = 20\%$$
$$500 * (2500 - 500) = 1{,}000{,}000$$

```
➢ a
5 25 1000
191 wins of 1000

➢ a
5 25 1000
203 wins of 1000

➢ a
500 2500 1000
197 wins of 1000
```

uses about 1 billion coin flips →

Remarks

- Computer simulation can help validate mathematical analysis.
- For this problem, mathematical analysis is simpler (if you know the math).
- For more complicated variants, computer simulation may be the best plan of attack.



Debugging

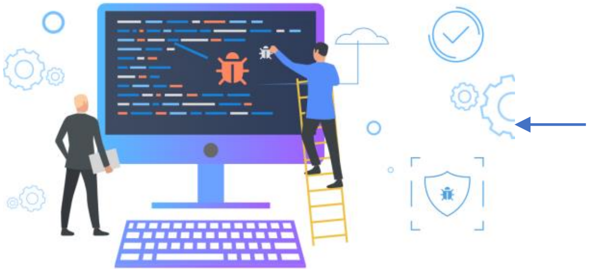
Eliminating mistakes from your programs

Debugging

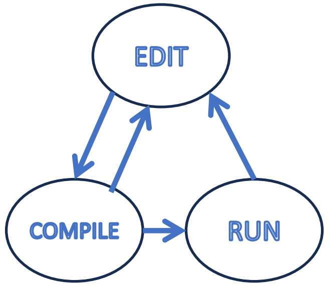
is 99% of program development in any programming language, *even for experts*.

Bug: A mistake in a program.

Debugging: The process of eliminating bugs.



You will make many mistakes as you write programs. It's normal.



Impossible ideal: "Please compile, execute, and debug my program."

Bottom line: Programming is primarily a process of finding and fixing mistakes.

Debugging

is challenging because conditionals and loops dramatically increase the number of possible outcomes.

program structure	no loops	n conditionals	1 loop
number of possible execution sequences	1	2^n	no limit

Most programs contain numerous conditionals and loops, with nesting.

Good news. Conditionals and loops provide structure that helps us understand our programs.

Debugging a program: a running example

Problem: Factor a large integer n.

Application: Cryptography.

Surprising fact: Security of internet commerce depends on difficulty of factoring large integers.

$$3{,}757{,}208 = 2 \times 2 \times 2 \times 7 \times 13 \times 13 \times 397$$

$$98 = 2 \times 7 \times 7$$

$$17 = 17$$

$$11{,}111{,}111{,}111{,}111{,}111 = 2{,}071{,}723 \times 5{,}363{,}222{,}357$$

Method

- Consider each integer i less than n
- While i divides n evenly

  Print i (it is a factor of n).

  Replace n with n/i .

Rationale:

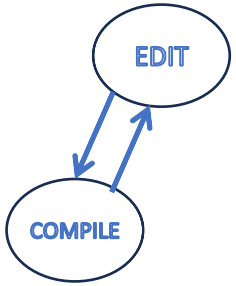
1. Any factor of n/i is a factor of n.
2. i may be a factor of n/i.

```
#include<iostream>
int main() {
    long long n;
    std::cin >> n;
    for (i = 0; i < n; i++)
    {
      while (n % i == 0)
        std::cout << i << " "
        n = n / i
    }
}
```

Debugging a program: syntax errors

Is your program a legal Java program?

- ○ C++ compiler can help you find out.
- ○ Find the first compiler error (if any).
- ○ Repeat.
- ○ Result: An executable a.exe file



```
➢ g++ factors.cpp
factors.cpp: In function 'int main()’:
factors.cpp:5:14: error: 'i' was not declared in this scope
```

```
➢ g++ factors.cpp
factors.cpp: In function 'int main()’:
factors.cpp:8:38: error: expected ';' before 'n'
```

```
➢ g++ factors.cpp
➢
```

```
#include<iostream>
int main() {
        long long n;
        std::cin >> n;
        for (int i = 0; i < n; i++)
        {
            while (n % i == 0)
                std::cout << i << " ";
                n = n / i;
        }
}
```

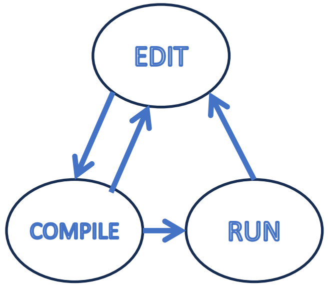
need to declare variable i

need terminating semicolons

Debugging a program: runtime and semantic errors

Does your legal C++ program do what you want it to do?

- ○ You need to run it to find out.
- ○ Find the first runtime error (if any).
- ○ Fix and repeat.



```
➤ ./a.out
98
Floating point exception
```

```
➤ ./a.out
98
2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2
. . .
```

```
➤ ./a.out
98
2 7 7 ➤
```

```
#include<iostream>
int main() {
        long long n;
        std::cin >> n;
        for (int i = 2; i < n; i++)
        {
            while (n % i == 0)
              { std::cout << i << " ";
                n = n / i; }
        }
}
```

need to start at 2 since 0 and 1 are not factors

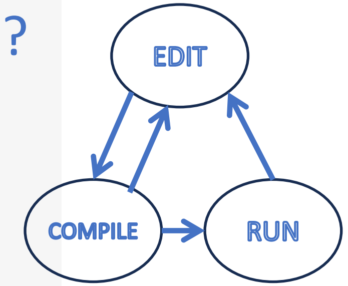
need braces

This working program still has bugs!

Debugging a program: testing

Does your legal Java program always do what you want it to do?

- ○ You need to test on many types of inputs it to find out.
- ○ Add trace code to find the first error.
- ○ Fix the error.
- ○ Repeat.



```
➢ ./a.out
98
2 7 7 ➢
```

need new line

```
➢ ./a.out
5

```

no output ??

```
➢ ./a.out
6
2
```

where is 3?

```
➢ ./a.out
5
TRACE 2 5
TRACE 3 5
TRACE 4 5
➢ ./.a.out
6
TRACE 2 3
```

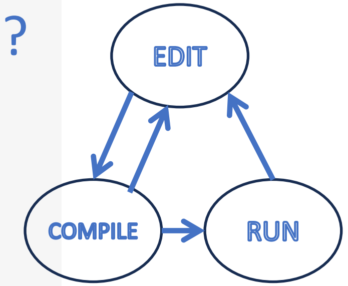
AHA! Need to print out n
(if it is not 1).

```
#include<iostream>
int main() {
    long long n;
    std::cin >> n;
    for (int i = 2; i < n; i++) {
        while (n % i == 0) {
            std::cout << i << " ";
            n = n / i;
        }
std::cout << "TRACE " << i << " " << n << "\n";
    }
}
```

Debugging a program: testing

Does your legal Java program always do what you want it to do?

- You need to test on many types of inputs it to find out.
- Add trace code to find the first error.
- Fix the error.
- Repeat.



```
➢ ./a.out
5
5
➢ ./.a.out
6
2 3
➢ ./.a.out
98
2 7 7
➢ ./aout
3757208
2 2 2 7 13 13 397
```

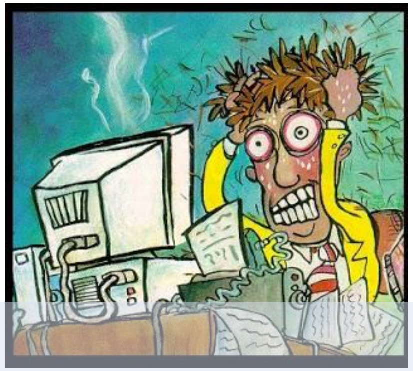
```
#include<iostream>
int main() {
    long long n;
    std::cin >> n;
    for (long long i = 2; i < n; i++) {
        while (n % i == 0) {
            std::cout << i << " ";
            n = n / i;
        }
    }
    if (n > 1) std::cout << n;
    std::cout << std::endl;
}
```

Note: This working program still has a bug (stay tuned).

Debugging a program: performance

Is your working Java program fast enough to solve your problem?

- You need to test it on increasing problem sizes to find out.
- May need to change the algorithm to fix it.
- Repeat.



```
➢ ./a.out
11111111
11 73 101 137
➢ ./a.out
11111111111
21649 513239
➢ ./a.out
11111111111111
11 239 4649 909091
➢ ./a.out
11111111111111111
2071723 5363222357
```

might work,
but way too slow

change the algorithm: no need to check when $i \cdot i > n$ since all smaller factors already checked

implement the change

```
#include<iostream>
int main() {
   long long n;
   std::cin >> n;
   for(long long i = 2; i <= n/i; i++) {
      while (n % i == 0) {
            std::cout << i << " ";
            n = n / i;
      }
    }
    if (n > 1) std::cout << n;
    std::cout << std::endl;
}
```

Debugging a program: performance analysis

Q. How large an integer can I factor?

digits in largest factor	$i < n$	$i \le \frac{n}{i}$
3	instant	instant
6	instant	instant
9	77 seconds	instant
12	21 hours §	instant
15	2.4 years §	2.7 seconds
18	2.4 millenia §	92 seconds

§ estimated, using analytic number theory

```
#include<iostream>
int main() {
    long long n;
    std::cin >> n;
    for (long long i = 2; i <= n/i; i++) {
        while (n % i == 0) {
            std::cout << i << " ";
            n = n / i;
        }
    }
    if (n > 1) std::cout << n;
    std::cout << std::endl;
}
```

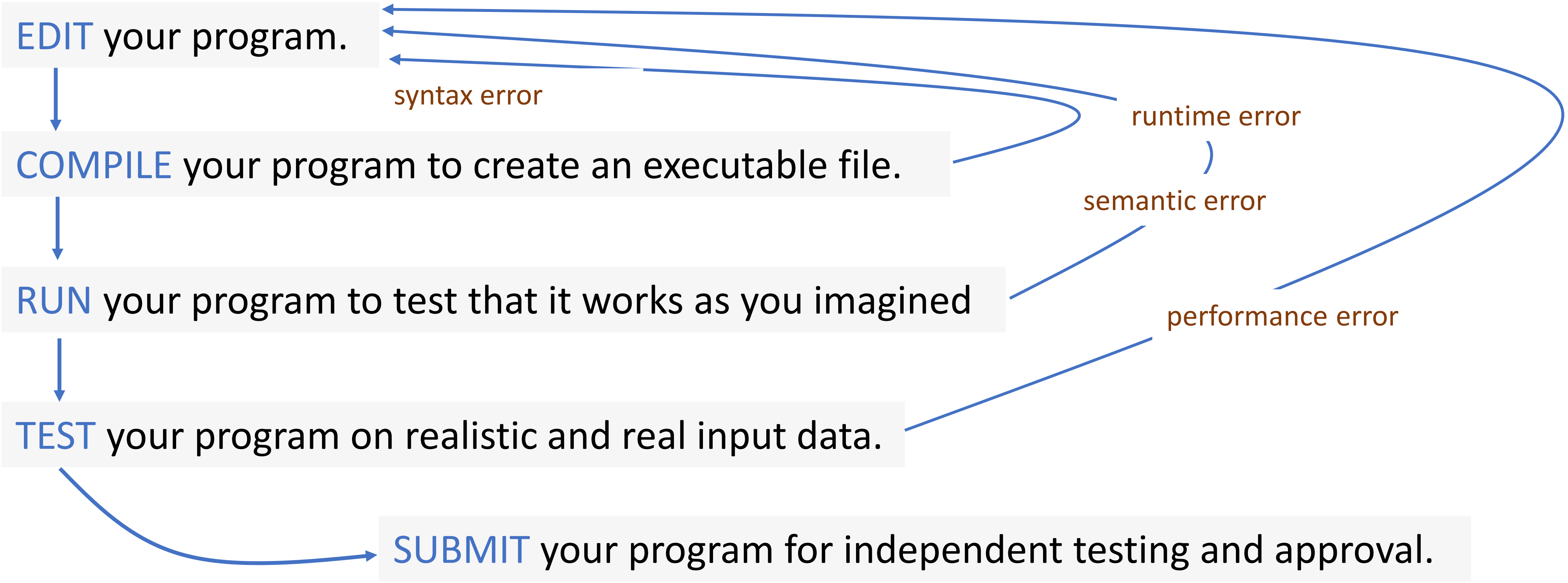
experts are still trying to develop better algorithms for this problem

Lesson. Performance matters!

Note. Internet commerce is still secure: it depends on the difficulty of factoring 200-digit integers.

Debugging your program: summary

Program development is a *four*-step process, with feedback.



EDIT your program.

syntax error

runtime error

semantic error

COMPILE your program to create an executable file.

RUN your program to test that it works as you imagined

performance error

TEST your program on realistic and real input data.

SUBMIT your program for independent testing and approval.

Other conditional and loop constructs

The `break` and `continue` statements, `switch` statement, `do-while` loops

The break statement

To exit a loop without letting it run to completion.

Two ways to leave the loop:

- Either the break statement is executed
  (because n is not prime)
- Or the loop-continuation condition is not satisfied
  (because n is prime).

```
#include<iostream>
int main() {
    long long n; std::cin >> n;
    bool isPrime = true;
    if (n < 2) isPrime = false;

    for (long long factor = 2; factor*factor <= n; factor++) {
        if (n % factor == 0) {
            isPrime = false;
            break;
        }
    }
    if (isPrime) std::cout << n << " is prime\n";
    else         std::cout << n << " is not prime\n";
}
```

The `continue` statement

To skip to the next iteration of a loop

If i is equals to 6 continue to next iteration without printing

```
#include<iostream>
int main() {
    for (int i = 1; i <= 10; i++) {
        if (i == 6)
            continue;
        std::cout << i << " ";
    }

    std::cout << std::endl;
}
```

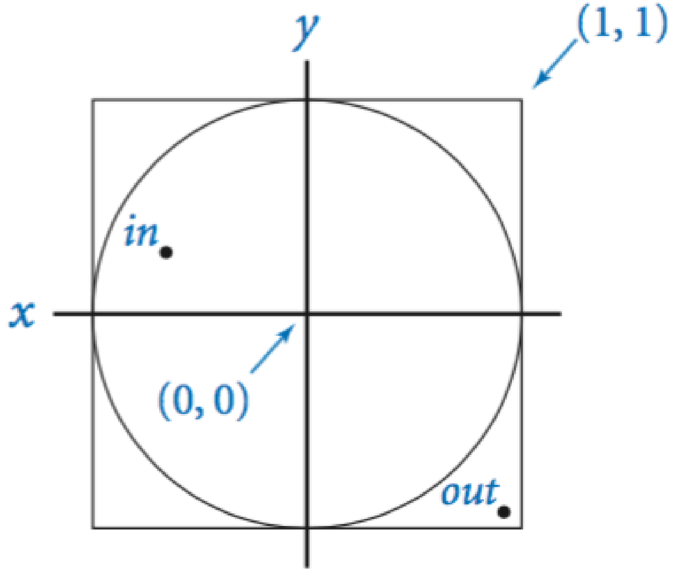
```
➢ ./a.out
1 2 3 4 5 7 8 9 10
```

The `do-while` loop

Same as a while loop except that the loop-continuation condition is omitted the first time through the loop.

Example: Sets x and y so that (x, y) is randomly distributed inside the circle centered at (0, 0) with radius 1.

```
#include<iostream>
#include<cstdlib>
int main() {
    srand(time(0));
    double x, y;
    do {
        x = rand()/(double)RAND_MAX;
        y = rand()/(double)RAND_MAX;
    } while(x*x+y*y > 1.0);
    std::cout << "(" << x << ", " << y << ")" << std::endl;
}
```



The `switch` statement

```
#include<iostream>

int main() {
    int d; std::cin >> d;

    if(d==0)
        std::cout << "Sunday";
    else if(d==1)
        std::cout << "Monday";
    else if(d==2)
        std::cout << "Tuesday";
    else if(d==3)
        std::cout << "Wednesday";
    else if(d==4)
        std::cout << "Thursday";
    else if(d==5)
        std::cout << "Friday";
    else if(d==6)
        std::cout << "Saturday";
    else
        std::cout << "Invalid day";
}
```

The `switch` statement

```
#include<iostream>
int main() {
    int d; std::cin >> d;
    switch(d) {
        case 0:
            std::cout << "Sunday";
            break;
        case 1:
            std::cout << "Monday";
            break;
        case 2:
            std::cout << "Tuesday";
            break;
        case 3:
            std::cout << "Wednesday";
            break;
        case 4:
            std::cout << "Thursday";
            break;
        case 5:
            std::cout << "Friday";
            break;
        case 6:
            std::cout << "Saturday";
            break;
        default:
            std::cout << "Invalid day";
            break;
    }
}
```