A Puzzle For You

Problem: Write a program to output the first $N$ cubes, but without using multiplication (only addition/subtraction).

Historically, CPUs are relatively slow at multiplication vs addition/subtraction.
- The differences can be small (3x) or large (20x).
The Correctness of Algorithms and Programs

CMPT 125
Mo Chen
SFU Computing Science
3/2/2020
Lecture 13

Today:

● Assertions and Invariants
● Good Invariants and Post-Conditions
● Proving Programs Correct
int main () {
    int N = 10;
    for (int i = 0; i < N; i++) {
        // Compute square = i*i
        int square = 0;
        for (int j = 0; j < i; j++) {
            // Assertion:  square == j*i
            square += i;
        }
        // Compute cube = i*i*i
        int cube = 0;
        for (int j = 0; j < i; j++) {
            cube += square;
        }
        printf("%d\n", cube);
    }
}

The Algorithm (Pseudocode):
For each $i$ from 0 to $N - 1$:
- Compute the $i$\textsuperscript{th} square by adding $i$ to itself $i$ times.
- Compute the $i$\textsuperscript{th} cube by adding the $i$\textsuperscript{th} square to itself $i$ times.
- Output the $i$\textsuperscript{th} cube.

Do you believe that at the end of this loop, the value of square will equal $i*i$?

Q. What’s a good assertion?

Good assertions, also called loop invariants, are usually related to the post-condition.
What makes a *good* loop invariant?

A loop invariant is a statement that is true every loop.

- usually asserted at the beginning of the loop
- usually parametrized by the loop index (\( j \) in this case)

A good loop invariant should indicate the progress of the algorithm

- the invariant should carry all state information, loop to loop.
- the invariant should imply the post-condition (the goal of the algorithm) at the end of the last loop.

```java
// Post: square == i*i
int square = 0;
for (int j = 0; j < i; j++) {
    // Assertion: square == j*i
    square += i;
}
```
Proving Correctness

Use mathematical reasoning to capture the behaviour of an algorithm:

- State *invariants* at various *checkpoints*.
- Show that the invariant holds:
  - at the first checkpoint
  - during execution between checkpoints
- Conclude that the post-condition holds
  - the invariant holds at / after the last checkpoint

```c
// Post:  square == i*i
int square = 0;
for (int j = 0; j < i; j++) {
    // Assertion:  square == j*i
    square += i;
}
```

- true when \( j = 0 \)
- if true for loop \( j \), then true for next \( j \)
- true when \( j = i \)
Proof

Base case:
- Is the invariant true on the first loop?
  - When \( j = 0 \), square has been initialized to 0. These values satisfy \( \text{square} = j \times i \).

Induction step:
- If the invariant holds at the beginning of loop \( j \), does it also hold for the beginning of loop \( j+1 \)?
  - At the beginning of loop \( j \), \( \text{square} = j \times i \).
  - After running the loop, \( \text{square} = j \times i + i = (j+1) \times i \), which is the invariant of the next loop.

Termination:
- Since the invariant holds for all \( j \), it holds after the last loop.
  - Therefore, when \( j = i \), \( \text{square} = i \times i \).
Yay - we proved it! So what?

We honestly won’t care whether or not you can do a proof of correctness 5 years from now in your job. And neither will:

- your boss
- your co-workers
- ♥♥♥ your secret crush ♥♥♥

You learn to do proofs to get better at reasoning about code.

The more practiced you are at thinking about invariants:

- the better your resulting code will be
- the easier it will be to figure out other people’s code

A computer won’t be able to verify your programs for you

- in general, this is an impossible problem.
What does it do?

```c
int main () {
    int N = 10;
    int a = 6;
    int b = 1;
    int c = 0;

    for (int i = 0; i < N; i++) {
        printf("%d\n", c);
        c += b;
        b += a;
        a += 6;
    }
}
```

Two ways to get started:
1. Simulate the execution on paper.
2. Key in the program and run it!

Output:

```
0
1
8
27
64
125
216
343
512
729
```

Q. Why is this program significant?
int main () {
    int N = 10;
    int a = 6;
    int b = 1;
    int c = 0;

    for (int i = 0; i < N; i++) {
        // Assertion: a = 6(i+1)
        // Assertion: b = 3i(i+1)(i+1) + 1
        // Assertion: c = i^3
        printf("%d\n", c);
        c += b;
        b += a;
        a += 6;
    }
    // Assertion: b = 3i(i+1)(i+1) + 1
    // Assertion: c = i^3
}

Since the assertion \( c = i^3 \) holds on every loop, the algorithm is correct.

**Base case:** When \( i = 0 \), the assertions are:
- \( a = 6(0+1) = 6 \)
- \( b = 3\cdot 0 \cdot (0+1) + 1 = 1 \)
- \( c = (0)^3 = 0 \)
which are the 3 initial values for \( a, b, c \).

**Induction step:** At the start of loop \( i \), the assertions are:
- \( a = 6(i+1) \)
- \( b = 3i(i+1) + 1 \)
- \( c = i^3 \)
After \( c += b \); the value of \( c \) changes to
- \( c = i^3 + 3i(i+1) + 1 \)
  = \( i^3 + 3i^2 + 3i + 1 \)
  = \((i+1)^3\)
After \( b += a \); the value of \( b \) changes to
- \( b = 3i(i+1) + 1 + 6(i+1) \)
  = \((i+1)(3i+6) + 1 \)
  = 3(i+1)(i+2) + 1
After \( a += 6 \); the value of \( a \) changes to
- \( a = 6(i+1) + 6 \)
  = \( 6(i+2) \)
which are the values for \( a, b, c \) on loop \( i + 1 \).
int main () {
    int N = 10;
    int a = 6;
    int b = 1;
    int c = 0;

    for (int i = 0; i < N; i++) {
        // Assertion: \( a = 6(i + 1) \)
        // Assertion: \( b = 3i(i + 1) + 1 \)
        // Assertion: \( c = i^3 \)
        printf("%d
", c);
        c += b;
        b += a;
        a += 6;
    }
}

Since the assertion \( c = i^3 \) holds on every loop, the algorithm is correct.

**Base case:** When \( i = 0 \), the assertions are:
- \( a = 6(0+1) = 6 \)
- \( b = 3 \cdot 0 \cdot (0+1) + 1 = 1 \)
- \( c = (0)^3 = 0 \)
which are the 3 initial values for \( a, b, c \).

**Induction step:** At the start of loop \( i \), the assertions are:
- \( a = 6(i + 1) \)
- \( b = 3i(i + 1) + 1 \)
- \( c = i^3 \)

After \( c += b; \) the value of \( c \) changes to
- \( c = i^3 + 3i(i + 1) + 1 \)
  \[ = i^3 + 3i^2 + 3i + 1 \]
  \[ = (i + 1)^3 \]

After \( b += a; \) the value of \( b \) changes to
- \( b = 3i(i + 1) + 1 + 6(i + 1) \)
  \[ = (i + 1)(3i + 6) + 1 \]
  \[ = 3(i + 1)(i + 2) + 1 \]

After \( a += 6; \) the value of \( a \) changes to
- \( a = 6(i + 1) + 6 \)
  \[ = 6(i + 2) \]
which are the values for \( a, b, c \) on loop \( i + 1 \).
Example: Reversing a String S

reverse(“stressed”) returns “desserts”
head(“stressed”) = “s”
tail(“stressed”) = “tressed”

X ← S; Y ← “”
while X not empty do:
    Y ← head(X) + Y
    X ← tail(X)
return Y

certainly for any nonempty S:
S = head(S) + tail(S)
Adding Invariants & Checkpoints

X <- S; Y <- ""
while X not empty do:
    Y <- head(X) + Y
    X <- tail(X)
return Y

What's a good invariant?
S = reverse(Y) + X

checkpoint 1: before the first loop
checkpoint 2: at the end of each loop
Proving the Invariant

\[
X \leftarrow S; \ Y \leftarrow "" \\
\text{chkpt 1: } // \ inv: \ S = \text{reverse}(Y) + X \\
\text{while } X \text{ not empty do:} \\
\quad Y \leftarrow \text{head}(X) + Y \\
\quad X \leftarrow \text{tail}(X) \\
\text{chkpt 2: } // \ inv: \ S = \text{reverse}(Y) + X \\
\text{return } Y
\]

Is it true at checkpoint 1?
Yes, because \(\text{reverse}(Y) + X = \text{reverse}("") + X = "" + X = X\)

Is it true at checkpoint 2?
Execute checkpoint to checkpoint: If the invariant is true at the beginning of the loop, then is it true at the end of the loop.
Suppose that $S = \text{reverse}(Y) + X$ at the beginning of some loop. Then after running the next loop $Y' = \text{head}(X) + Y$ and $X' = \text{last}(X)$. So, just need to show that $S = \text{reverse}(Y') + X'$. \\
$\text{reverse}(Y') + X' = \text{reverse}(\text{head}(X) + Y) + \text{last}(X) = \text{reverse}(Y) + \text{head}(X) + \text{last}(X) = \text{reverse}(Y) + X = S$. 
One Last Detail

So, the invariant holds at the beginning of the first loops, and at the end of every successive loop. Including the last loop!

- When $X = "\"$, the loop terminates and $S = \text{reverse}(Y) + X = \text{reverse}(Y)$. Thus $Y = \text{reverse}(S)$.

But does the loop terminate?

- Another invariant: that $|X|$ is natural and decreasing.