Introduction to Programming (in C++)

*Algorithms on sequences.*

*Reasoning about loops: Invariants.*

Jordi Cortadella, Ricard Gavaldà, Fernando Orejas
Dept. of Computer Science, UPC
Outline

• Algorithms on sequences
  – Treat-all algorithms
  – Search algorithms

• Reasoning about loops: invariants
Maximum of a sequence

• Write a program that tells the largest number in a non-empty sequence of integers.

// Pre: a non-empty sequence of integers is ready to be read at cin
// Post: the maximum number of the sequence has been written at the output

Assume the input sequence is: 23 12 -16 34 25

<table>
<thead>
<tr>
<th>elem:</th>
<th>-</th>
<th>12</th>
<th>-16</th>
<th>34</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>m:</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>34</td>
<td>34</td>
</tr>
</tbody>
</table>

// Invariant: m is the largest number read from the sequence
Maximum of a sequence

```c
int main() {
    int m;
    int elem;
    cin >> m;
    // Inv: m is the largest element read
    //      from the sequence
    while (cin >> elem) {
        if (elem > m) m = elem;
    }
    cout << m << endl;
}
```

Why is this necessary?
Checks for end-of-sequence and reads a new element.
Reading with `cin`

• The statement `cin >> n` can also be treated as a Boolean expression:
  – It returns `true` if the operation was successful
  – It returns `false` if the operation failed:
    • no more data were available (EOF condition) or
    • the data were not formatted correctly (e.g. trying to read a double when the input is a string)

• The statement:

```cpp
cin >> n
```

can be used to detect the end of the sequence and read a new element simultaneously. If the end of the sequence is detected, `n` is not modified.
Finding a number greater than N

• Write a program that detects whether a sequence of integers contains a number greater than N.

// Pre: at the input there is a non-empty sequence of integers in which the first number is N.
// Post: writes a Boolean value that indicates whether a number larger than N exists in the sequence.

Assume the input sequence is: 23 12 -16 24 25

<table>
<thead>
<tr>
<th>num:</th>
<th>-</th>
<th>12</th>
<th>-16</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>found:</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

// Invariant: “found” indicates that a value greater than N has been found.
Finding a number greater than N

```cpp
int main() {
    int N, num;
    cin >> N;
    bool found = false;

    // Inv: found indicates that a number greater than N has been found
    while (not found and cin >> num) {
        found = num > N;
    }
    cout << found << endl;
}
```
Algorithmic schemes on sequences

• The previous examples perform two different operations on a sequence of integers:
  – Finding the maximum number
  – Finding whether there is a number greater than N

• They have a distinctive property:
  – The former requires all elements to be visited
  – The latter requires one element to be found
Treat-all algorithms

- A classical scheme for algorithms that need to treat all the elements in a sequence

```
Initialize (the sequence and the treatment)
// Inv: The visited elements have been treated
while (not end of sequence) {
    Get a new element e;
    Treat e;
}
```
Search algorithms

- A classical scheme for algorithms that need to find an element with a certain property in a sequence.

```cpp
bool found = false;
Initialize;
// Inv: “found” indicates whether the element has been found in the visited part of the sequence
while (not found and not end of sequence) {
    Get a new element e;
    if (Property(e)) found = true;
}
// “found” indicates whether the element has been found.
// “e” contains the element.
```
Longest repeated subsequence

• Assume we have a sequence of strings

  **cat** dog bird cat bird bird cat cat cat cat dog mouse horse

• We want to calculate the length of the longest sequence of repetitions of the first string. Formally, if we have a sequence of strings

  $S_1, S_2, \ldots, S_n$

we want to calculate

$$\max\{j - i + 1 : 1 \leq i \leq j \leq n \land s_i = s_{i+1} = \cdots = s_{j-1} = s_j = s_1\}.$$
// Specification: see previous slide
// Variable to store the first string.
string first;
cin >> first;
string next; // Visited string in the sequence
// Length of the current and longest subsequences
int length = 1, longest = 1;

// Inv: "length" is the length of the current subsequence.
// "longest" is the length of the longest subsequence visited so far.
while (cin >> next) {
    if (first != next) length = 0; // New subsequence
    else {
        // The current one is longer
        length = length + 1;
        if (length > longest) longest = length;
    }
}

// "longest" has the length of the longest subsequence
Search in the dictionary

• Assume we have a sequence of strings representing words. The first string is a word that we want to find in the dictionary that is represented by the rest of the strings. The dictionary is ordered alphabetically.

• Examples:

  dog ant bird cat cow dog eagle fox lion mouse pig rabbit shark whale yak

  frog ant bird cat cow dog eagle fox lion mouse pig rabbit shark whale yak

• We want to write a program that tells us whether the first word is in the dictionary or not.
Search in the dictionary

// Specification: see previous slide
// First word in the sequence (to be sought).
string word;
cin >> word;

// A variable to detect the end of the search
// (when a word is found that is not smaller than “word”).
bool found = false;

// Visited word in the dictionary (initialized as empty for
// the case in which the dictionary might be empty).
string next = "";

// Inv: not found => the visited words are smaller than “word”
while (not found and cin >> next) found = next >= word;
// “found” has detected that there is no need to read the rest of
// the dictionary
found = word == next;
// “found” indicates that the word was found.
Increasing number

• We have a natural number $n$. We want to know whether its representation in base 10 is a sequence of increasing digits.

• Examples:

  134679  → increasing
  56688  → increasing
  3  → increasing
  134729  → non-increasing
Increasing number (I)

// Pre: n >= 0
// Returns true if the sequence of digits representing n (in base 10)
// is increasing, and returns false otherwise

bool increasing(int n) {
    // The algorithm visits the digits from LSB to MSB.
    bool incr = true;
    int previous = 9; // Stores a previous “fake” digit

    // Inv: n contains the digits no yet treated, previous contains the
    //      last treated digit (that can never be greater than 9),
    //      incr implies all the treated digits form an increasing sequence
    while (incr and n > 0) {
        int next = n%10;
        incr = next <= previous;
        previous = next;
        n /= 10;
    }

    return incr;
}
// Pre: n >= 0
// Returns true if the sequence of digits representing n (in base 10)
// is increasing, and returns false otherwise

bool increasing(int n) {
    // The algorithm visits the digits from LSB to MSB.
    int previous = 9; // Stores a previous “fake” digit

    // Inv: n contains the digits no yet treated, previous contains the
    //      last treated digit (that can never be greater than 9) and
    //      all the previously treated digits form an increasing sequence
    while (n > 0) {
        int next = n%10;
        if (next > previous) return false;
        previous = next;
        n /= 10;
    }

    return true;
}

**Exercise**: write the function increasing(int n, int b) with the same specification, but for a number representation in base b.
Insert a number in an ordered sequence

• Read a sequence of integers that are all in ascending order, except the first one. Write the same sequence with the first element in its correct position.

• Note: the sequence has at least one number. The output sequence must have a space between each pair of numbers, but not before the first one or after the last one.

• Example

Input: 15 2 6 9 12 18 20 35 75
Output: 2 6 9 12 15 18 20 35 75

• The program can be designed with a combination of search and treat-all algorithms.
int first;
cin >> first;

bool found = false; // controls the search of the location
int next; // the next element in the sequence

// Inv: All the read elements that are smaller than the first have been written
// not found => no number greater than or equal to the first has been
// found yet
while (not found and cin >> next) {
    if (next >= first) found = true;
    else cout << next << " ";
}

cout << first;

if (found) {
    cout << " " << next;
    // Inv: all the previous numbers have been written
    while (cin >> next) cout << " " << next;
}
cout << endl;
• We have a sequence of characters representing a text that ends with ‘.’
• We want to calculate the number of words in the text.
• A word is a sequence of letters. Words are separated by characters that are not letters.
• There could be an undefined number of separators before the first word and after the last word.
• Example: the text

¡¡ Today is Friday !! Alice, are we going out for dinner? .

has 10 words.
Counting words

// Pre: the input contains a sequence of characters that ends with ‘.’

// Post: the number of words in the sequence has been written at the output.
char read_word():

// Pre: the last char from the input was a letter
// Returns the next char that is not a letter.

char next_word(char c):

// Pre: c is the last non-letter char read from the input.
// Returns the next char that is a letter or a ‘.’

Note: next_word will return c if it is a ‘.’
int main() {

    char c = next_word(' '); // A fake ' ' as parameter

    int count = 0;

    // Inv: count is the number of words in the treated part
    // of the input. c contains the first char in the
    // non-treated part
    while (c != '.') {
        count = count + 1;
        c = read_word();
        c = next_word(c);
        // or: c = next_word(read_word());
    }
    cout << count << endl;
}
// Pre: c is the last non-letter char read from the input.
// Returns the next char that is a letter or a ‘.’

char next_word(char c) {
    bool found = false;

    // Inv: found indicates that a letter has been found.
    //      c is the last char read from the input
    while (c != '.' and not found){
        c = cin.get();
        found = is_letter(c);
        /* cin.get() returns the next char at the input.
           (cin>>c skips spaces and other separating chars) */
    }

    return c;
}
// Pre: c is the last non-letter char read from the input.
// Returns the next char that is a letter or a ‘.’

char next_word(char c) {
    // a simpler solution
    // Inv: c is the last char read from the input
    while (c != '.' && !is_letter(c)) c = cin.get();
    return c;
}
/ Pre: the last char from the input was a letter
// Returns the next char that is not a letter.

char read_word() {
    char c = cin.get(); // Next char after the first letter
    // Inv: c is the last char read from the input
    while (is_letter(c)) c = cin.get();
    return c;
}
// Returns whether c is a letter

bool is_letter(char c) {
    return ('a' <= c and c <= 'z') or ('A' <= c and c <= 'Z');
}
REASONING ABOUT LOOPS: INVARIANTS
Invariants

• Invariants help to ...
  – Define how variables must be initialized before a loop
  – Define the necessary condition to reach the post-condition
  – Define the body of the loop
  – Detect whether a loop terminates

• It is crucial, but not always easy, to choose a good invariant.

• Recommendation:
  – Use invariant-based reasoning for all loops (possibly in an informal way)
  – Use formal invariant-based reasoning for non-trivial loops
General reasoning for loops

Initialization;

// Invariant: a proposition that holds
// * at the beginning of the loop
// * at the beginning of each iteration
// * at the end of the loop

// Invariant
while (condition) {
    // Invariant \land condition
    Body of the loop;
    // Invariant
}

// Invariant \land \neg condition
Example with invariants

- Given $n \geq 0$, calculate $n!$

- Definition of factorial:
  
  \[ n! = 1 \times 2 \times 3 \times \ldots \times (n-1) \times n \]

  (particular case: $0! = 1$)

- Let’s pick an invariant:
  - At each iteration we will calculate $f = i!$
  - We also know that $i \leq n$ at all iterations
Calculating n!

// Pre: n ≥ 0
// Returns n!
int factorial(int n) {
    int i = 0;
    int f = 1;
    // Invariant: f = i! and i ≤ n
    while (i < n) {
        // f = i! and i < n
        i = i + 1;
        f = f*i;
        // f = i! and i ≤ n
    }
    // f = i! and i ≤ n and i ≥ n
    // f = n!
    return f;
}
Reversing digits

• Write a function that reverses the digits of a number (representation in base 10)

• Examples:

  35276 → 67253
  19 → 91
  3 → 3
  0 → 0
Reversing digits

// Pre: n ≥ 0
// Returns n with reversed digits (base 10)

int reverse_digits(int n) {
    int r;

    r = 0;
    // Invariant (graphical): →
    while (n > 0) {
        r = 10*r + n%10;
        n = n/10;
    }

    return r;
}
Calculating $\pi$

- $\pi$ can be calculated using the following series:

$$\frac{\pi}{2} = 1 + \frac{1}{3} + \frac{1 \cdot 2}{3 \cdot 5} + \frac{1 \cdot 2 \cdot 3}{3 \cdot 5 \cdot 7} + \cdots$$

- Since an infinite sum cannot be computed, it may often be sufficient to compute the sum with a finite number of terms.
Calculating $\pi$

// Pre: nterms > 0
// Returns an estimation of $\pi$ using nterms terms
// of the series

double Pi(int nterms) {
    double sum = 1; // Approximation of $\pi/2$
    double term = 1; // Current term of the sum

    // Inv: sum is an approximation of $\pi/2$ with k terms,
    //      term is the k-th term of the series.
    for (int k = 1; k < nterms; ++k) {
        term = term * k / (2.0 * k + 1.0);
        sum = sum + term;
    }
    return 2 * sum;
}
Calculating $\pi$

- $\pi = 3.14159265358979323846264338327950288...$
- The series approximation:

<table>
<thead>
<tr>
<th>nterms</th>
<th>$\text{Pi(nterms)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.000000</td>
</tr>
<tr>
<td>5</td>
<td>3.098413</td>
</tr>
<tr>
<td>10</td>
<td>3.140578</td>
</tr>
<tr>
<td>15</td>
<td>3.141566</td>
</tr>
<tr>
<td>20</td>
<td>3.141592</td>
</tr>
<tr>
<td>25</td>
<td>3.141593</td>
</tr>
</tbody>
</table>