Introduction to Programming (in C++)

Advanced Sequence Processing

Lluís Padró
Dept. of Computer Science, UPC

Outline

- Sliding window strategy: Processing sequence elements that depend on neighbors
  - Treat-all algorithms
  - Search algorithms

- Sequences of sequences
  - Treat-all sequences, treat-all elements in each.
  - Search sequence, treat-all elements in each.
  - Search sequence, search element in each.
  - Treat-all sequences, search element in each.

Sliding Window Strategy

Sliding Window Strategy

- Write a program that counts the number of consecutive ascending pairs in a non-empty sequence of integers.

```
// Pre:  a non-empty sequence of integers is
//       ready to be read at cin
// Post: the number of ascending intervals from one element
//        to the next has been written to the output
```

Assume the input sequence is: 3 12 8 19 25 15

elem:	3	12	8	19	25	15
count:	0	1	1	2	3	3

```
// Invariant: m is the count of ascending intervals found
//            so far in the sequence.
```

Sliding Window Strategy

- Keep a “window” that checks two consecutive elements, and slides one position at a time:

3	12	8	19	25	15	Iteration 1. count = 1
3	12	8	19	25	15	Iteration 2. count = 1
3	12	8	19	25	15	Iteration 3. count = 2
3	12	8	19	25	15	Iteration 4. count = 3
3	12	8	19	25	15	Iteration 5. count = 3

Sliding Window Strategy

- The "window" can be emulated with two variables, one containint the current value, and another containing the previous value.

3	12	8	19	25	15
prev	curr				

```
Iteration 1.  prev=3, curr=12
```

Sliding Window Strategy

- The “window” can be emulated with two variables, one containint the current value, and another containing the previous value.

3	12	8	19	25	15
	prev	curr			

`Iteration 2.  prev=12, curr=8`

Sliding Window Strategy

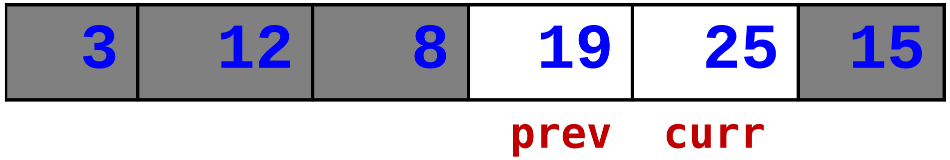
- The “window” can be emulated with two variables, one containint the current value, and another containing the previous value.

3	12	8	19	25	15
		prev	curr		

```
Iteration 3.  prev=8, curr=19
```

Sliding Window Strategy

- The "window" can be emulated with two variables, one containint the current value, and another containing the previous value.




```
Iteration 4.  prev=19, curr=25
```

Sliding Window Strategy

- The “window” can be emulated with two variables, one containint the current value, and another containing the previous value.

3	12	8	19	25	15
			prev	curr	

```
Iteration 4.  prev=19, curr=25
```

ETC...

Sliding Window Strategy

- We use a normal treat-all algorithm, but we introduce a new variable to keep the value of the previous element.

- Special attention must be paid to initialization:

  What is the element previous to the first ?

Count ascending consecutive pairs

```
int main() {
    int c = 0; // ascending pair counter

    int elem; // current element
    int prev; // previous element
    cin >> prev;
    while (cin >> elem) {
        // if ascending pair, count it.
        if (elem > prev) c = c + 1;
        // prepare for next iteration
        prev = elem;
    }
    cout << c << endl;
}
```

Sliding Window Strategy

- Windows may be of any size (2, 3, 4, …)
- We can do searches as well as treat-all algorithms.

Sliding Window Strategy

Write a program that checks whether a sequence of characters ending with a dot contains the combination “`hola`”.

`fgsdholasfgg.`

Sliding Window Strategy

- We need a window of size 4 (i.e. 4 variables: 1 for current character, 3 for previous elements)
- Search algorithm: If the combination is found, there is no need to keep checking.

```
fgsdholasfgg.
fgsdholasfgg.
fgsdholasfgg.
fgsdholasfgg.
fgsdholasfgg.
```

Sliding Window Strategy

- 4 variables for the window: a,b,c,d
- Advance one position at a time

```
fgsdholasfgg.
a b c d
```

Sliding Window Strategy

- 4 variables for the window: a,b,c,d
- Advance one position at a time

fgsdholasfgg.

a b c d

Sliding Window Strategy

- 4 variables for the window: a,b,c,d
- Advance one position at a time

```
fgsdholasfgg.
  a b c d
```

Sliding Window Strategy

- 4 variables for the window: a,b,c,d
- Advance one position at a time

```
fgsdholasfgg.
   a b c d
```

Sliding Window Strategy

- 4 variables for the window: a,b,c,d
- Advance one position at a time

```
fgsdholasfgg.
    a b c d
```

Sliding Window Strategy

- We use a normal search algorithm, but we introduce **three** new variables to keep the value of the previous elements.

- Special attention must be paid to initialization:
  What are the 3 elements previous to the first ?

The sequence may have less than 4 characters!!

Find ‘hola’ in a sequence of characters

```
int main() {
    char a,b,c; // 3 previous elements
    char d;     // current element
    // init previous elements to something inocuous
    a=‘_‘; b=‘_‘; c=‘_‘;
    cin >> d;
    bool found = false;
    while (not found and d != ‘.’) {
        found = (a==’h’ and b==’o’ and
                 c==’l’ and d==’a’)
        // prepare for next iteration
        a = b; b = c; c = d;
        cin >> d;
    }
    if (found) cout << “yes” << endl;
    else  cout << “no” << endl;
}
```

Sliding Window Strategy

- Similar problems:
  - Compute the maximum difference between one element and the next in a sequence of integers. (treat-all, window=2)
  - Compute length of longest sequence of consecutive repetitions of the same word. (treat-all, window=2)
  - Find out whether a sequence of integers is ascending. (search, window=2)
  - Compute maximum ‘peak’ in a sequence of integers (treat-all, window=3)

Sequences of Sequences

Sequences of sequences

- Single process sequence is applied to a collection of sequences
- Example: Given a several sequences of integers, each ended in zero, compute the maximum of each sequence.

Input

```
3 5 8 1 10 4 9 0
12 5 6 1 7 0
9 22 31 1 1 5 0
1 0
```

Output

```
10
12
31
1
```

Sequences of sequences

- Single process sequence is applied to a collection of sequences plus an overall computation.
- Example: Given a several sequences of integers, each ended in zero, compute the maximum of each sequence and the sum of the maximums.

Input

```
3 5 8 1 10 4 9 0
12 5 6 1 7 0
9 22 31 1 1 5 0
1 0
```

Output

```
10
12
31
1
Sum=54
```

Sequences of sequences

- When dealing with sequences of sequences, two things must be taken into account:
  - Task Structure
    - We check all sequences or we stop when a certain sequence is found ?
    - Inside each sequence, we check all elements, or we stop when a certain element is found ?
  - Input Structure
    - Each sequence ends with a marking element or we know the number of elements it contains?
    - We read sequences until there are no more, or we know the number of sequences to read?

Sequences of sequences

- Task Structure:
  - We may need to check all sequences (treat-all sequences), or stop when one with certain properties is found (sequence search)
  - Inside each sequence, we may need to check all elements (treat-all elements), or stop when one with certain properties is found (element search)

4 possible combinations

Sequences of sequences: Task Structure

- 4 possible combinations:
  1. Treat-all sequences, treat-all elements
  2. Search sequence, treat-all elements
  3. Search sequence, search element
  4. Treat-all sequences, search elements

Sequences of sequences: Task Structure

- Example problems:
  - Given several sequences of integers, count the average amount of prime numbers per sequence (treat-all sequences, treat-all elements).
  - Given several sequences of char, check whether one of the sequences contains the char combination “hola” an even number of times (search sequence, treat-all elements).
  - Given several sequences of integers, find out which is the first sequence that contains a prime number (search sequence, search element).
  - Given several sequences of integers, output the first position in each sequence that contains a prime number (treat-all sequences, search element)

Sequences of sequences: Task Structure

- 4 possible combinations:
  1. Treat-all sequences, treat-all elements
  2. Search sequence, treat-all elements
  3. Search sequence, search element
  4. Treat-all sequences, search elements

1. Treat-all sequences, treat-all elements

Example: Compute the maximum of each sequence, and the sum of all maximums.

Each sequence ends in zero and has at least one element.

```
12 10 8 7 5 0
1  22 0
4  0
3  -4 1 0
```

```
int main() {
    int sum = 0;
    int x;
    while (cin >> x) {
        int m = x;
        while (x != 0) {
            if (x > m) m = x;
            cin >> x
        }
        cout << m << endl;
        sum = sum + m;
    }
    cout << sum << endl;
}
```

1. Treat-all sequences, treat-all elements

Main loop reads the first element of each sequence, until there are no more sequences.

Each iteration processes a whole sequence.

```
int main() {
    int sum = 0;
    int x;
    while (cin >> x) {
        Compute maximum of current
        sequence, until zero is found.
        Store result in m.
        cout << m << endl;
        sum = sum + m;
    }
    cout << sum << endl;
}
```

1. Treat-all sequences, treat-all elements

Inner loop reads the rest of elements of each sequence, until the marking zero is found.

Each iteration processes one element and stores the maximum of seen elements.

```
int main() {
    int sum = 0;
    int x;
    while (cin >> x) {
        int m = x;
        while (x != 0) {
          if (x > m) m = x;
          cin >> x
        }
        cout << m << endl;
        sum = sum + m;
    }
    cout << sum << endl;
}
```

Sequences of sequences: Task Structure

- 4 possible combinations:
  1. Treat-all sequences, treat-all elements
  2. Search sequence, treat-all elements
  3. Search sequence, search element
  4. Treat-all sequences, search elements

2. Search sequence, treat-all elements

Example: Check if any of the sequences sums over 50.

Each sequence ends in zero.

```
12  10  -7  5  0
1   22  0
4   0
3   -4  1  0
```

```
int main() {
    bool found = false;
    int x;
    while (cin >> x and not found) {
        int s = 0;
        while (x != 0) {
            s = s + x;
            cin >> x;
        }
        found = (s > 50);
    }
    if (found) cout << “yes” << endl;
    else cout << “no” << endl;
}
```

2. Search sequence, treat-all elements

Main loop reads the first element of each sequence, until there are no more sequences, or a matching sequence is found

Each iteration processes a whole sequence.

```
int main() {
    bool found = false;
    int x;
    while (cin >> x and not found) {
```

Compute sum of current sequence, until zero is found. Store result in s.

```
        }
        found = (s > 50);
    }
    if (found) cout << “yes” << endl;
    else cout << “no” << endl;
}
```

2. Search sequence, treat-all elements

Inner loop reads the rest of elements of each sequence, until the marking zero is found.

Each iteration processes one element and accumulates the sum of seen elements.

```
int main() {
    bool found = false;
    int x;
    while (cin >> x and not found) {
        int s = 0;
        while (x != 0) {
            s = s + x;
            cin >> x;
        }
        found = (s > 50);
    }
    if (found) cout << "yes" << endl;
    else cout << "no" << endl;
}
```

Sequences of sequences: Task Structure

- 4 possible combinations:
  1. Treat-all sequences, treat-all elements
  2. Search sequence, treat-all elements
  3. Search sequence, search element
  4. Treat-all sequences, search elements

3. Search sequence, search element

Example: Locate the first sequence that contains a number ending in 3.

Each sequence ends in zero.

```
12 10 7 5 0
1  22 0
4  0
3  4 1 0
```

```
int main() {
    bool found = false;
    int x;
    int p = 0;
    while (cin >> x and not found) {
        bool end3 = false;
        while (x != 0 and not end3) {
            end3 = (x%10 == 3);
            cin >> x;
        }
        found = end3;
        p = p + 1;
    }
    if (found) cout << p << endl;
    else cout << “none” << endl;
}
```

3. Search sequence, search element

Main loop reads the first element of each sequence, until there are no more sequences, or a matching sequence is found

Each iteration processes a whole sequence.

```
int main() {
    bool found = false;
    int x;
    int p = 0;
    while (cin >> x and not found) {
```

Check if current sequence contains a prime number
Store result in ‘end3’.

```
        found = end3;
        p = p + 1;
    }
    if (found) cout << p << endl;
    else cout << “none” << endl;
}
```

3. Search sequence, search element

Inner loop reads the rest of elements of each sequence, until the marking zero is reached or a number ending in 3 is found.

Each iteration processes one element .

```
int main() {
    bool found = false;
    int x;
    int p = 0;
    while (cin >> x and not found) {
        bool end3 = false;
        while (x != 0 and not end3) {
            end3 = (x%10 == 3);
            cin >> x;
        }
        found = end3;
        p = p + 1;
    }
    if (found) cout << p << endl;
    else cout << “none” << endl;
}
```

Sequences of sequences: Task Structure

- 4 possible combinations:
  1. Treat-all sequences, treat-all elements
  2. Search sequence, treat-all elements
  3. Search sequence, search element
  4. Treat-all sequences, search elements

4. Treat-all sequences, search element

Example: Count how many sequences contain a multiple of 10.

Each sequence ends in zero.

```
12 10 -7 5 0
1  22 0
4  0
3  -4 1 0
```

```
int main() {
    int n = 0;
    int x;
    while (cin >> x) {
        bool mult = false;
        while (x != 0) {
            if (x%10 == 0)
                mult = true;
            cin >> x;
        }
        if (mult) n = n + 1;
    }
    cout << n << endl;
}
```

4. Treat-all sequences, search element

Main loop reads the first element of each sequence, until there are no more sequences.

Each iteration processes a whole sequence.

```
int main() {
    int n = 0;
    int x;
    while (cin >> x) {

        if (mult) n = n + 1;
    }
    cout << n << endl;
}
```

Check if current sequence contains a multiple of 10.
Store result in mult.

4. Treat-all sequences, search element

Inner loop reads the rest of elements of each sequence, until the marking zero is reached, checking if a multiple of 10 is found.

Each iteration processes one element .

```
int main() {
    int n = 0;
    int x;
    while (cin >> x) {
        bool mult = false;
        while (x != 0) {
            if (x%10 == 0)
                mult = true;
            cin >> x;
        }
        if (mult) n = n + 1;
    }
    cout << n << endl;
}
```

4. Treat-all sequences, search element

Inner loop reads the rest of elements of each sequence, until the marking zero is reached, checking if a multiple of 10 is

```
int main() {
    int n = 0;
    int x;
    while (cin >> x) {
        bool mult = false;
        while (x != 0) {
            if (x%10 == 0)
                mult = true;
            cin >> x;
}
```

BUT we can NOT stop when we find a multiple of 10 !!
We MUST read all elements until the 0, to keep in sync with the main loop.

Sequences of sequences

- Input Structure
  - We may be given the number of sequences, have an end mark, or we may need to read as many as they come.
  - Inside each sequence, we may be given the number of elements, or the end of the sequence may be identified with a marker element.

6 possible combinations

Sequences of sequences: Input Structure

- 6 possible combinations:
  1. Known number of sequences, known number of elements in each.
  2. Known number of sequences, final mark in each.
  3. Unknown number of sequences, known number of elements in each.
  4. Unknown number of sequences, final mark in each.
  5. Mark indicating no more sequences, known number of elements in each.
  6. Mark indicating no more sequences, final mark in each.

Sequences of sequences: Input Structure

6 possible combinations:

1. Known number of sequences, known number of elements in each.
2. Known number of sequences, final mark in each.
3. Unknown number of sequences, known number of elements in each.
4. Unknown number of sequences, final mark in each.
5. Mark indicating no more sequences, known number of elements in each.
6. Mark indicating no more sequences, final mark in each.

1. Known number of sequences, known number of elements in each

```
4
5  12 10 8 7 5
1  22
0
3  0  -4  1
```

1. Known number of sequences, known number of elements in each

Number of sequences

```
4
5  12 10 8 7 5
1  22
0
3  0  -4  1
```

1. Known number of sequences, known number of elements in each

Number of sequences

Number of elements in each sequence

```
4
5  12 10 8 7 5
1  22
0
3  0  -4  1
```

1. Known number of sequences, known number of elements in each

Number of sequences

Number of elements in each sequence

```
4
5  12 10 8 7 5
1  22
0
3  0  -4  1
```

Elements in each sequence

1. Known number of sequences, known number of elements in each

The actual position of the elements does not matter

```
4 5 12 10 8 7 5 1  22 0 3  0  -4  1
```

1. Known number of sequences, known number of elements in each

The actual position of the elements does not matter

```
4 5 12 10 8 7 5 1  22 0 3  0  -4  1
```

Number of sequences

1. Known number of sequences, known number of elements in each

The actual position of the elements does not matter

```
4 5 12 10 8 7 5 1  22 0 3  0  -4  1
```

Number of elements in seq 1

Elements of seq 1

1. Known number of sequences, known number of elements in each

The actual position of the elements does not matter

```
4 5 12 10 8 7 5 1 22 0 3  0  -4  1
```

Number of elements in seq 2

Elements of seq 2

1. Known number of sequences, known number of elements in each

The actual position of the elements does not matter

```
4 5 12 10 8 7 5 1   22 0 3   0   -4   1
```

Number of elements in seq 3

1. Known number of sequences, known number of elements in each

The actual position of the elements does not matter

```
4 5 12 10 8 7 5 1  22 0 3 0  -4  1
```

Number of elements in seq 4

Elements of seq 4

1. Known number of sequences, known number of elements in each

```
// get number of sequences
int ns;
cin >> ns;

for (int i=0; i<ns; ++i) {
    // get number of elements in sequence #i
    int ne;
    cin >> ne

    for (int j=0; j<ne; ++j) {
        // get element #j in seq #i
        int x;
        cin >> x;

        // process element
    }
}
```

Sequences of sequences: Input Structure

6 possible combinations:

1. Known number of sequences, known number of elements in each.
2. Known number of sequences, final mark in each.
3. Unknown number of sequences, known number of elements in each.
4. Unknown number of sequences, final mark in each.
5. Mark indicating no more sequences, known number of elements in each.
6. Mark indicating no more sequences, final mark in each.

2. Known number of sequences, final mark in each

```
4
12 10 8 7 5 0
1  22 0
0
3  -4  1 0
```

2. Known number of sequences, final mark in each

Number of sequences

```
4
12 10 8 7 5 0
1  22 0
0
3  -4  1 0
```

2. Known number of sequences, final mark in each

Number of sequences

```
4
12 10 8 7 5 0
1  22 0
0
3  -4  1 0
```

Elements in each sequence

2. Known number of sequences, final mark in each

Number of sequences

```
4
12 10 8 7 5 0
1  22 0
0
3  -4  1 0
```

Final mark in each sequence

Elements in each sequence

2. Known number of sequences, final mark in each

Again, the actual position of the elements does not matter

```
4 12 10 8 7 5 0 1 22 0 0 3 -4 1 0
```

2. Known number of sequences, final mark in each

Again, the actual position of the elements does not matter

```
4 12 10 8 7 5 0 1 22 0 0 3 -4 1 0
```

Number of sequences

2. Known number of sequences, final mark in each

Again, the actual position of the elements does not matter

```
4 12 10 8 7 5 0 1 22 0 0 3 -4 1 0
```

Number of sequences

Elements of seq 1

End mark of seq 1

2. Known number of sequences, final mark in each

Again, the actual position of the elements does not matter

```
4 12 10 8 7 5 0 1 22 0 0 3 -4 1 0
```

Number of sequences

Elements of seq 3

End mark of seq 2

2. Known number of sequences, final mark in each

Again, the actual position of the elements does not matter

```
4 12 10 8 7 5 0 1 22 0 0 3 -4 1 0
```

Number of sequences

End mark of seq 3 (no elements)

2. Known number of sequences, final mark in each

Again, the actual position of the elements does not matter

```
4 12 10 8 7 5 0 1 22 0 0 3 -4 1 0
```

Number of sequences

Elements of seq 4

End mark of seq 4

2. Known number of sequences, final mark in each

```
// get number of sequences
int ns;
cin >> ns;

for (int i=0; i<ns; ++i) {
   // get elements in sequence until mark is found
   int x;
   cin >> x
   while (x != 0) {
      // process element x

      cin >> x;
   }
}
```

Sequences of sequences: Input Structure

6 possible combinations:

1. Known number of sequences, known number of elements in each.
2. Known number of sequences, final mark in each.
3. Unknown number of sequences, known number of elements in each.
4. Unknown number of sequences, final mark in each.
5. Mark indicating no more sequences, known number of elements in each.
6. Mark indicating no more sequences, final mark in each.

3. Unknown number of sequences, known number of elements in each

```
5 12 10 8 7 5
2 1  22
0
3 2 -4  1
```

3. Unknown number of sequences, known number of elements in each

Number of elements in each sequence

```
5 12 10 8 7 5
2 1  22
0
3 2 -4  1
```

3. Unknown number of sequences, known number of elements in each

Number of elements in each sequence

```
5 12 10 8 7 5
2 1  22
0
3 2 -4  1
```

Elements in each sequence

3. Unknown number of sequences, known number of elements in each

Again, the actual position of the elements does not matter

```
5 12 10 8 7 5 2 1 22 0 3 2 -4 1
```

3. Unknown number of sequences, known number of elements in each

Again, the actual position of the elements does not matter

```
5 12 10 8 7 5 2 1 22 0 3 2 -4 1
```

Number of elements in seq 1

Elements of seq 1

3. Unknown number of sequences, known number of elements in each

Again, the actual position of the elements does not matter

```
5 12 10 8 7 5 2 1 22 0 3 2 -4 1
```

Number of elements in seq 2

Elements of seq 2

3. Unknown number of sequences, known number of elements in each

Again, the actual position of the elements does not matter

```
5 12 10 8 7 5 2 1 22 0 3 2 -4 1
```

Number of elements
in seq 3 (no elements)

3. Unknown number of sequences, known number of elements in each

Again, the actual position of the elements does not matter

```
5 12 10 8 7 5 2 1 22 0 3 2 -4 1
```

Number of elements in seq 4

Elements of seq 4

3. Unknown number of sequences, known number of elements in each

```
// get number of elements of each sequence (if any)
int ne;
while (cin >> ne) {

    for (int j=0; j<ne; ++j) {
        // get element #j in current sequence
        int x;
        cin >> x;

        // process element
    }
}
```

Sequences of sequences: Input Structure

6 possible combinations:

1. Known number of sequences, known number of elements in each.
2. Known number of sequences, final mark in each.
3. Unknown number of sequences, known number of elements in each.
4. Unknown number of sequences, final mark in each.
5. Mark indicating no more sequences, known number of elements in each.
6. Mark indicating no more sequences, final mark in each.

4. Unknown number of sequences, final mark in each

```
12 10 8 7 5 0
1  22 0
0
3  -4  1 0
```

4. Unknown number of sequences, final mark in each

```
12 10 8 7 5 0
1  22 0
0
3  -4  1 0
```

Elements in each sequence

4. Unknown number of sequences, final mark in each

```
12 10 8 7 5 0
1  22 0
0
3  -4  1 0
```

Final mark in each sequence

Elements in each sequence

4. Unknown number of sequences, final mark in each

Again, the actual position of the elements does not matter

```
12 10 8 7 5 0 1 22 0 0 3 -4 1 0
```

4. Unknown number of sequences, final mark in each

Again, the actual position of the elements does not matter

```
12 10 8 7 5 0 1 22 0 0 3 -4 1 0
```

Elements of seq 1

End mark of seq 1

4. Unknown number of sequences, final mark in each

Again, the actual position of the elements does not matter

```
12 10 8 7 5 0 1 22 0 0 3 -4 1 0
```

Elements of seq 2

End mark of seq 3

4. Unknown number of sequences, final mark in each

Again, the actual position of the elements does not matter

```
12 10 8 7 5 0 1 22 0 0 3 -4 1 0
```

End mark of
seq 3
(no elements)

4. Unknown number of sequences, final mark in each

Again, the actual position of the elements does not matter

```
12 10 8 7 5 0 1 22 0 0 3 -4 1 0
```

Elements of seq 4

End mark of seq 4

4. Unknown number of sequences, final mark in each

```
// get first element of each
// sequence (if any)
int x;
while (cin >> x) {

   // get elements in sequence
   // until mark is found
   while (x != 0) {
      // process element x

      cin >> x;
   }
}
```

Sequences of sequences: Input Structure

6 possible combinations:

1. Known number of sequences, known number of elements in each.
2. Known number of sequences, final mark in each.
3. Unknown number of sequences, known number of elements in each.
4. Unknown number of sequences, final mark in each.
5. Mark indicating no more sequences, known number of elements in each.
6. Mark indicating no more sequences, final mark in each.

5. Mark indicating no more sequences, known number of elements in each.

```
5 12 10 8 7 5
2 1  22
0
3 3 -4  1
-1
```

5. Mark indicating no more sequences, known number of elements in each.

Number of elements in each sequence

```
5 12 10 8 7 5
2 1  22
0
3 3 -4  1
-1
```

5. Mark indicating no more sequences, known number of elements in each.

```
5 12 10 8 7 5
2 1  22
0
3 3 -4  1
-1
```

Elements in each sequence

5. Mark indicating no more sequences, known number of elements in each.

```
5 12 10 8 7 5
2 1  22
0
3 3 -4  1
-1
```

Final mark (no more sequences)

5. Mark indicating no more sequences, known number of elements in each.

Again, the actual position of the elements does not matter

```
5 12 10 8 7 5 2 1 22 0 3 3 -4 1 -1
```

Again, the actual position of the elements does not matter

```
5 12 10 8 7 5 2 1 22 0 3 3 -4 1 -1
```

Number of elements in seq 1

Elements of seq 1

5. Mark indicating no more sequences, known number of elements in each.

Again, the actual position of the elements does not matter

```
5 12 10 8 7 5 2 1 22 0 3 3 -4 1 -1
```

Number of elements in seq 2

Elements of seq 2

Again, the actual position of the elements does not matter

```
5 12 10 8 7 5 2 1 22 0 3 3 -4 1 -1
```

Number of elements
in seq 3 (no elements)

5. Mark indicating no more sequences, known number of elements in each.

Again, the actual position of the elements does not matter

```
5 12 10 8 7 5 2 1 22 0 3 3 -4 1 -1
```

Number of elements in seq 4

Elements of seq 4

5. Mark indicating no more sequences, known number of elements in each.

Again, the actual position of the elements does not matter

```
5 12 10 8 7 5 2 1 22 0 3 3 -4 1 -1
```

Final mark
(no more sequences)

5. Mark indicating no more sequences, known number of elements in each.

```
// get first element of each
// sequence (which may be the mark)
int ne;
cin >> ne;
while (ne != -1) {
    // get elements in sequence,
    // as many as 'ne' indicates
    for (int i=0; i<ne; ++i) {
        int x;
        cin >> x;
        // process element x
    }
    // number of elements of next
    // sequence (or final mark)
    cin >> ne;
}
```

Sequences of sequences: Input Structure

- 6 possible combinations:
  1. Known number of sequences, known number of elements in each.
  2. Known number of sequences, final mark in each.
  3. Unknown number of sequences, known number of elements in each.
  4. Unknown number of sequences, final mark in each.
  5. Mark indicating no more sequences, known number of elements in each.
  6. Mark indicating no more sequences, final mark in each.

6. Mark indicating no more sequences, final mark in each.

```
12 10 8 7 5 0
1  22 0
0
3  -4  1 0
-1
```

6. Mark indicating no more sequences, final mark in each.

```
12 10 8 7 5 0
1  22 0
0
3  -4  1 0
-1
```

Elements in each sequence

6. Mark indicating no more sequences, final mark in each.

```
12 10 8 7 5 0
1  22 0
0
3  -4  1 0
-1
```

Final mark in each sequence

Elements in each sequence

6. Mark indicating no more sequences, final mark in each.

```
12 10 8 7 5 0
1  22 0
0
3  -4  1 0
-1
```

Final mark, (no more sequences)

6. Mark indicating no more sequences, final mark in each.

Again, the actual position of the elements does not matter

```
12 10 8 7 5 0 1 22 0 0 3 -4 1 0 -1
```

6. Mark indicating no more sequences, final mark in each.

Again, the actual position of the elements does not matter

```
12 10 8 7 5 0 1 22 0 0 3 -4 1 0 -1
```

Elements of seq 1

End mark of seq 1

6. Mark indicating no more sequences, final mark in each.

Again, the actual position of the elements does not matter

```
12 10 8 7 5 0 1 22 0 0 3 -4 1 0 -1
```

Elements of seq 2

End mark of seq 3

6. Mark indicating no more sequences, final mark in each.

Again, the actual position of the elements does not matter

```
12 10 8 7 5 0 1 22 0 0 3 -4 1 0 -1
```

End mark of
seq 3
(no elements)

6. Mark indicating no more sequences, final mark in each.

Again, the actual position of the elements does not matter

```
12 10 8 7 5 0 1 22 0 0 3 -4 1 0 -1
```

Elements of seq 4

End mark of seq 4

6. Mark indicating no more sequences, final mark in each.

Again, the actual position of the elements does not matter

```
12 10 8 7 5 0 1 22 0 0 3 -4 1 0 -1
```

Final mark (no more sequences)

6. Mark indicating no more sequences, final mark in each.

```
// get first element of each
// sequence (which may be the mark)
int x;
cin >> x;
while (x != -1) {
   // get elements in sequence
   // until mark is found
   while (x != 0) {
      // process element x
      cin >> x;
   }
   // first element of next
   // sequence (or the final mark)
   cin >> x;
}
```