Introduction to Programming (in C++)

Advanced examples

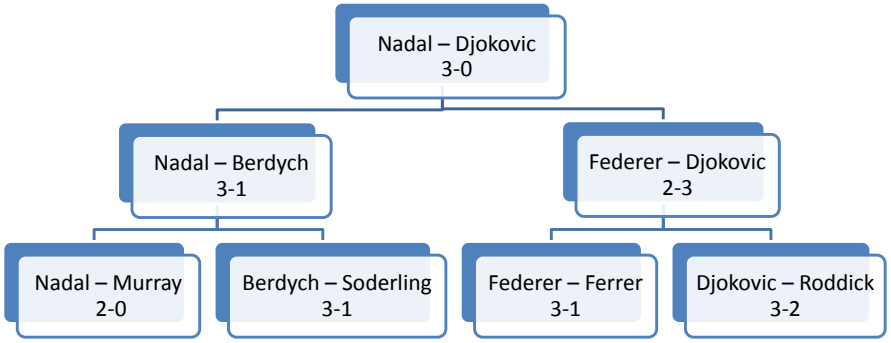
Jordi Cortadella, Ricard Gavaldà, Fernando Orejas
Dept. of Computer Science, UPC

Sports tournament

- Design a program that reads the participants in a knockout tournament and the list of results for each round. The program must write the name of the winner.
- Assumptions:
  - The number of participants is a power of two.
  - The list represents the participation order, i.e. in the first round, the first participant plays with the second, the third with the fourth, etc. In the second round, the winner of the first match plays against the winner of the second match, the winner of the third match plays against the winner of the fourth match, etc. At the end, the winner of the first semi-final will play against the winner of the second semi-final.
- The specification of the program could be as follows:

```
// Pre:  the input contains the number of players,
//       the players and the results of the tournament.
// Post: the winner has been written at the output.
```


Sports tournament



- Input (example):

```
8 Nadal Murray Berdych Soderling Federer
Ferrer Djokovic Roddick
2 0 3 1 3 1 3 2 3 1 2 3 3 0
```


Sports tournament

- A convenient data structure that would enable an efficient solution would be a vector with 2*n*-1 locations (*n* is the number of participants):
  - The first *n* locations would store the participants.
  - The following *n*/2 locations would store the winners of the first round.
  - The following *n*/4 locations would store the winners of the second round, etc.
  - The last location would store the name of the winner.

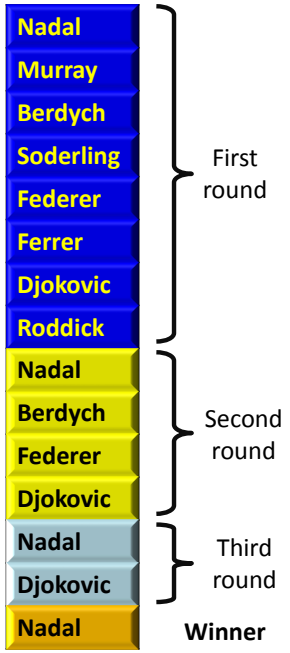
Sports tournament

- Input:

```
8

Nadal Murray Berdych
Soderling Federer Ferrer
Djokovic Roddick

2 0 3 1 3 1 3 2 3 1 2 3 3 0
```

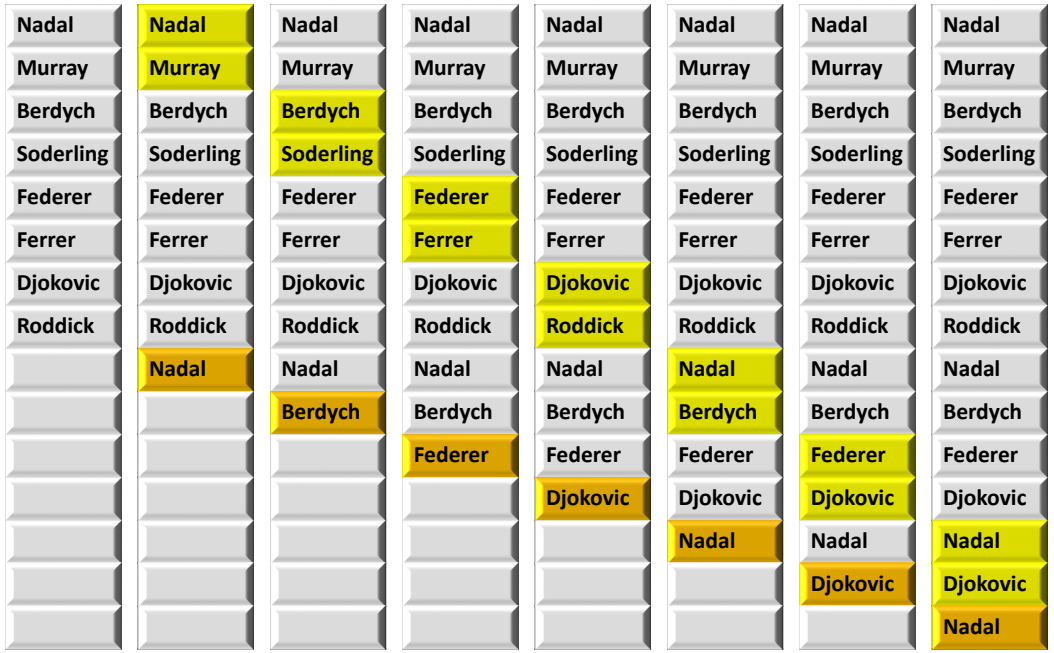


Sports tournament

- The algorithm could run as follows:
  - First, it reads the number of participants and their names. They will be stored in the locations 0…*n*-1 of the vector.
  - Next, it fills up the rest of the locations. Two pointers might be used. The first pointer (j) points at the locations of the players of a match. The second pointer (k) points at the location where the winner will be stored.

```
// Inv: players[n..k-1] contains the
//      winners of the matches stored
//      in players[0..j-1]
```


Sports tournament



Sports tournament

```
int main() {
    int n;
    cin >> n; // Number of participants
    vector<string> players(2*n - 1);
    // Read the participants
    for (int i = 0; i < n; ++i) cin >> players[i];
    int j = 0;
    // Read the results and calculate the winners
    for (int k = n; k < 2*n - 1; ++k) {
        int score1, score2;
        cin >> score1 >> score2;
        if (score1 > score2) players[k] = players[j];
        else players[k] = players[j + 1];
        j = j + 2;
    }
    cout << players[2*n - 2] << endl;
}
```

Sports tournament

- Exercise:

  Modify the previous algorithm using only a vector with n strings, i.e.,

```
vector<string> players(n)
```


Permutations

- Given a number N, generate all permutations of the numbers 1…N in lexicographical order.

For N=4:

1 2 3 4	2 1 3 4	3 1 2 4	4 1 2 3
1 2 4 3	2 1 4 3	3 1 4 2	4 1 3 2
1 3 2 4	2 3 1 4	3 2 1 4	4 2 1 3
1 3 4 2	2 3 4 1	3 2 4 1	4 2 3 1
1 4 2 3	2 4 1 3	3 4 1 2	4 3 1 2
1 4 3 2	2 4 3 1	3 4 2 1	4 3 2 1

Permutations

```
// Structure to represent the prefix of a permutation.
// When all the elements are used, the permutation is
// complete.
// Note: used[i] represents the element i+1

struct Permut {
    vector<int> v; // stores a partial permutation (prefix)
    vector<bool> used; // elements used in v
};
```

Permutations

```
void BuildPermutation(Permut& P, int i);

// Pre:  P.v[0..i-1] contains a prefix of the permutation.
//       P.used indicates the elements present in P.v[0..i-1]
// Post: All the permutations with prefix P.v[0..i-1] have
//       been printed in lexicographical order.
```

Permutations

```
void BuildPermutation(Permut& P, int i) {
    if (i == P.v.size()) {
        PrintPermutation(P); // permutation completed
    } else {
        // Define one more location for the prefix
        // preserving the lexicographical order of
        // the unused elements
        for (int k = 0; k < P.used.size(); ++k) {
            if (not P.used[k]) {
                P.v[i] = k + 1;
                P.used[k] = true;
                BuildPermutation(P, i + 1);
                P.used[k] = false;
            }
        }
    }
}
```

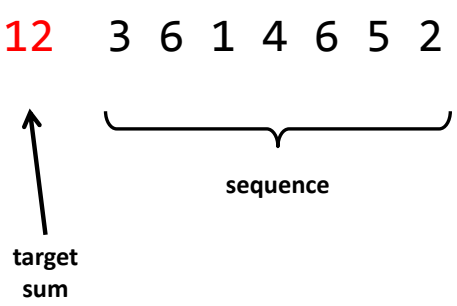

Permutations

```
int main() {
    int n;
    cin >> n; // will generate permutations of {1..n}
    Permut P; // creates a permutation with empty prefix
    P.v = vector<int>(n);
    P.used = vector<bool>(n, false);
    BuildPermutation(P, 0);
}

void PrintPermutation(const Permut& P) {
    int last = P.v.size() - 1;
    for (int i = 0; i < last; ++i) cout << P.v[i] << " ";
    cout << P.v[last] << endl;
}
```


Sub-sequences summing *n*

- Given a sequence of positive numbers, write all the sub-sequences that sum another given number *n*.
- The input will first indicate the target sum. Next, all the elements in the sequence will follow, e.g.



Sub-sequences summing *n*

```
> 12 3 6 1 4 6 5 2
3 6 1 2
3 1 6 2
3 4 5
6 1 5
6 4 2
6 6
1 4 5 2
1 6 5
4 6 2
```

Sub-sequences summing *n*

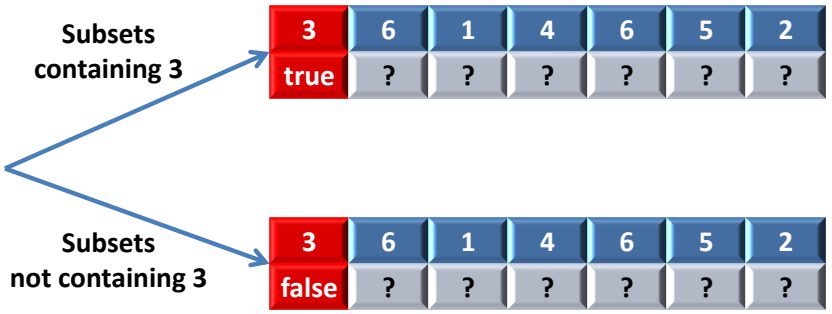
- How do we represent a subset of the elements of a vector?
  - A Boolean vector can be associated to indicate which elements belong to the subset.

Value:	3	6	1	4	6	5	2
Chosen:	false	true	true	false	false	true	false

represents the subset **{6,1,5}**

Sub-sequences summing *n*

- How do we generate all subsets of the elements of a vector? Recursively.
  - Decide whether the first element must be present or not.
  - Generate all subsets with the rest of the elements



Sub-sequences summing *n*

- How do we generate all the subsets that sum *n*?
  - Pick the first element (3) and generate all the subsets that sum *n-3* starting from the second element.

3	6	1	4	6	5	2
true	?	?	?	?	?	?

  - Do not pick the first element, and generate all the subsets that sum *n* starting from the second element.

3	6	1	4	6	5	2
false	?	?	?	?	?	?

Sub-sequences summing *n*

```
struct Subset {
    vector<int> values;
    vector<bool> chosen;
};

void main() {
    // Read number of elements and sum
    int sum;
    cin >> sum;

    // Read sequence
    Subset s;
    int v;
    while (cin >> v) {
      s.values.push_back(v);
      s.chosen.push_back(false);
    }
    // Generates all subsets from element 0
    generate_subsets(s, 0, sum);
}
```

Sub-sequences summing *n*

```
void generate_subsets(Subset& s, int i, int sum);

// Pre:  s.values is a vector of n positive values and
//       s.chosen[0..i-1] defines a partial subset.
//       s.chosen[i..n-1] is false.

// Post: A list of subsets has been printed. The subsets
//       agree with s.chosen[0..i-1] such that the sum of
//       the chosen values in s.values[i..n-1] is sum.


// Terminal cases:
//   · sum < 0 → nothing to print
//   · sum = 0 → print the subset
//   · i >= n  → nothing to print
```


Sub-sequences summing *n*

```
void generate_subsets(Subset& s, int i, int sum) {
    if (sum >= 0) {
        if (sum == 0) print_subset(s);
        else if (i < s.values.size()) {

            // Recursive case: pick i and subtract from sum
            s.chosen[i] = true;
            generate_subsets(s, i + 1, sum - s.values[i]);

            // Do not pick i and maintain the sum
            s.chosen[i] = false;
            generate_subsets(s, i + 1, sum);
        }
    }
}
```


Sub-sequences summing *n*

```
// Pre:  s.values contains a set of values and
//       s.chosen indicates the values to be printed
// Post: the chosen values have been printed in cout

void print_subset(const Subset& s) {
    for (int i = 0; i < s.values.size(); ++i) {
        if (s.chosen[i]) cout << s.values[i] << " ";
    }
    cout << endl;
}
```