Exercise 8

TEMPLATES AND STL

- Templates
  - implement one generic p2m method for *any* kernel
- C++ STL
  - vector, sort, …

Exercise 8

Question 1: Templatized P2M method

- implement a standalone p2m method
  - avoid redundant code by using templates
  - remove previous p2m member methods
  - only kernel-specific members in kernel classes

- Two possibilities
  - either pass an object of each kernel to p2m
  - or make members of each kernel class *static*

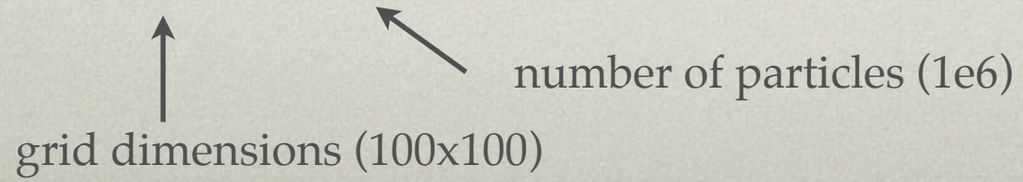
Exercise 8 installation

- Starting code is the solution of Exercise 7
  - Download the zip from the course website
  - Unzip the folder Exercise8

- Compile the code
  - move to the makefile directory and type:

```
make clean
make
```

- Run the code
  - ./p2m 100 1000000



Exercise 8
Question 2: C++ STL

What is the output?

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  int main (int argc, char * const argv[]) {
7      vector<int> v;
8
9      for (int i=0; i<10; ++i) v.push_back(i);
10
11     vector<int>::reverse_iterator it;
12     for (it = v.rbegin(); it != v.rend(); it++)
13         cout << *it << " ";
14
15     cout << endl;
16
17     return 0;
18 }
```

Exercise 8
Question 3: C++ STL

What is the output?

```
1  #include <vector>
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8      vector<int> v(3,2);
9
10     for (int i=1; i<=3; i++)
11         v.push_back(i);
12
13     for (int i=0; i<v.size(); i++)
14         cout << v[i] << " ";
15
16     return 0;
17 }
```

Use an iterator

Exercise 8

Question 4: C++ STL and Templates

Templatize print_vec

What is the output?

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

void print_vec(const std::vector<int> &myvector)
{
  for (std::vector<int>::const_iterator it=myvector.begin();
    it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';
}

int main () {

  int myints[] = {32,71,53,45,26,80,54,33};

  std::vector<int> myvector(myints, myints+8);

  print_vec(myvector);

  std::sort (myvector.begin()+2, myvector.end()-1);

  print_vec(myvector);

  std::vector<float> myvector2(2, 1.234);

  print_vec(myvector2);

  return 0;
}
```