Sequence and array parameters

```
/*
    . Sequences and arrays
        . Passing arrays as function parameters
        . Passing (seemingly) sized arrays as function parameters: The size is ignored by the compiler
        . Passing arrays by reference: Enforce the size
        . Passing std::vector or std::array as function parameters
*/
```

array as a function parameter

```cpp
//Passing array function parameters
// double sum( double scores[], size_t count){
double sum(double *scores, size_t count)
{

    double score_sum{};

    fmt::println( "sizeof(scores)  : {}" , sizeof(scores) );    //Inside the function the array decays to a pointer
    fmt::println( "sizeof(int*) : {}" , sizeof (int *) );
    //auto arr_size = std::size(scores);  // std::size doesn't work here because the array has decayed to a pointer

    for (size_t i{ 0 }; i < count; ++i) {
            score_sum += scores[i];
    }
    return score_sum;
}
int main(){

    //Passing array function parameters
    double my_scores[]{ 10.5, 34.3, 4.8, 6.5 };
    fmt::println( "sizeof(my_scores) : {}" , sizeof(my_scores) );


    double result = sum(my_scores, std::size(my_scores));
    fmt::println("result : {}", result);
}
```

(Seemingly) sized array parameter

```cpp
//Passing (seemingly) sized array function parameters
//The 5 size is not enforced. It's just ignored by the compiler.
double sum(double scores[5], size_t count)
{
    double sum{};

    for (size_t i{}; i < count; ++i) {
        sum += scores[i];
    }
    return sum;
}

int main(){
    //Passing (seemingly) sized array function parameters
    double student_scores[]{ 10.0, 20.0, 30.0, 40.0, 50.0, 60.0 };// Less than 5 parameters

    double result = sum(student_scores, std::size(student_scores));
    fmt::println("result : {}", result);
}
```

Enforce the size

```cpp
//Enforce the size. Pass the sized array by reference
//This is just a piece of syntax you have to remember.
//If the size is different from the one specified, the compiler will throw an error.
double sum(const double (&scores)[10])
{
  double sum{};
  for (size_t i{}; i < std::size(scores); ++i) { sum += scores[i]; }
  return sum;
}

int main(){

  //Enforce the size. Pass the sized array by reference
  double student_scores[]{ 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0};
  double sum_result = sum(student_scores);
  fmt::println("result is: {}", sum_result);

}
```

Multi-dimensional: declaration

```cpp
// Passing multi-dimensional arrays as function parameters (2d and 3d)
double sum(const double array[][3], size_t size)
{
  double sum{};
  for (size_t i{}; i < size; ++i)// Loop through rows
  {
    for (size_t j{}; j < 3; ++j)// Loop through elements in a row
    {
      sum += array[i][j];// Array access notation
      // sum += *( *(array + i) +j);  // Pointer arithmetic notation.// Confusing . Prefer array noation
    }
  }
  return sum;
}

double sum_3d(const double array[][3][2], size_t size)
{
  double sum{};
  for (size_t i{}; i < size; ++i)// Loop through rows
  {
    for (size_t j{}; j < 3; ++j)// Loop through elements in a row
    {
      for (size_t k{}; k < 2; ++k) {
        sum += array[i][j][k];  // sum += *(*(*(array + i) + j)+k);
      }
    }
  }
  return sum;
}
```

Multi-dimensional: use in main

```
//Passing multi-dimensional arrays as function parameters (2d and 3d)
double weights[][3]{
    { 10.0, 20.0,30.0},
    { 40.0, 50.0, 60.0 },
    { 70.0, 80.0, 90.0 },
    { 100.0, 110.0, 120.0 }
};

double weights_3d[][3][2]{
    {
        { 10, 20 },
        { 30, 40 },
        { 50, 60 },
    },
    {
        { 70, 80 },
        { 90, 100 },
        { 110, 120 },
    }
};

double result = sum(weights, std::size(weights));
fmt::println("2d array sum: {}", result);


result = sum_3d(weights_3d, std::size(weights_3d));
fmt::println("3d array sum: {}", result);
```

std::array as parameter

```cpp
//Passing std::array as a function parameter
double sum(const std::array<double, 10>& scores)
{
  double sum{};
  for (size_t i{}; i < scores.size(); ++i) { sum += scores[i]; }
  return sum;
}

int main(){
    //Passing std::array as a function parameter
    std::array<double, 10> student_scores{ 10.0, 20.0, 30.0};
    double sum_result = sum(student_scores);
    fmt::println("result is : {}", sum_result);

}
```