==========

## ARRAYS

==========

- **Introduction to arrays**
  - An array is used to process a collection of data of the same type
    - Examples:
      - a list of temperatures
      - a list of names
  - Why do we need to use arrays?
    - ...? (YOUR ANSWER) .

- **Declaring arrays and accessing array elements**
  - An array to store the final scores (of type `int` ) of the 196 PROG students:

```
int score[196];  // NOTE: the contents is undetermined
```

  - This is like declaring 196 variables of type `int`:

```
score[0], score[1], ... score[195]
```

  - The value in brackets is called a **subscript** OR an **index**
  - The variables making up the array are referred to as
    - indexed variables
    - subscripted variables
    - elements of the array
  - NOTE:
    - the first index value is zero
    - the largest index is one less than the size
  - Good practice:
    - Use constants to declare the size of an array:
      - using a constant allows your code to be easily altered for use on a smaller or larger set of data:

```
const int NUMBER_OF_STUDENTS = 196;
...
int score[NUMBER_OF_STUDENTS];
```

  - **NOTE**:
    - In C++, variable length arrays are not legal.

```
cout << "Enter number of students: ";
cin >> number;
int score[number];  // ILLEGAL IN MANY COMPILERS
```

    - G++ compiler allows this as an "extension" (because C99 allows it)

- **How arrays are stored in memory**
  - Declaring the array `int a[6]`
    - reserves memory for six variables of type **int**;
    - the variables are stored one after another
  - The address of `a[0]` is remembered
    - The addresses of the other indexed variables is not remembered
  - To determine the address of `a[3]` the compiler
    - starts at `a[0]`
    - counts past enough memory for three integers to find `a[3]`
  - VERY IMPORTANT NOTE:
    - A common error is using a nonexistent index.
    - Index values for `int a[6]` are the values `0` through `5` .
    - An index value not allowed by the array declaration is out of range.
    - Using an out of range index value does not necessarily produce an error message!!!

- **Initializing arrays**
  - Initialization when it is declared
    - `int a[3] = {11, 19, 12};`
    - `int a[ ] = {3, 8, 7, 1}; //size not needed`
    - `int b[100] = {0}; //all elements equal to zero`
    - `int b[5] = {4 ,7 ,9}; //4th & 5th elements equal to zero`
    - `string name[3] = {"Ana", "Rui", "Pedro"};`

  - Initialization after declaration

```
const int NUMBER_OF_STUDENTS = 196;
...
int score[NUMBER_OF_STUDENTS];
...
for (int i=0; i<NUMBER_OF_STUDENTS; i++)
     score[i] = 20;  // :-)
```

- **Operations on arrays**
  - It is <u>not possible</u> to copy all the array elements using a single assignment operation
  - It is <u>not possible</u> to read/write/process all the array elements with a "simple" statement

```
int a1[3] = {10,20,30};
int a2[3];
a2 = a1; // COMPILATION ERROR ...
cout << a1 << endl; //NOT AN ERROR! BUT ... WHAT DOES IT SHOW?
```

  - Each element must be read/written/processed at a time, using a loop (see previous example)

- **Arrays as function arguments / parameters & as return values**
  - Arrays can be arguments to functions.
  - A formal parameter can be for an entire array
    - such a parameter is called an array parameter
    - array parameters behave much like call-by-reference parameters
  - An array parameter is indicated using empty brackets in the parameter list
  - The number of array elements (to be processed) must be indicated as an additional formal parameter (`numStudents`, in the example below)

```
void readScores(int score[], int numStudents)
{
   ... // loop for reading student scores
}

...

int studentScore[NUMBER_OF_STUDENTS];
...

readScores(studentScore, NUMBER_OF_STUDENTS); //function call
```

  - **`Const` modifier**
    - Array parameters allow a function to change the values stored in the array argument (BE CAREFUL!)
    - If a function should not change the values of the array argument, use the modifier **`const`**

```
double computeScoreAverage(const int score[], int numStudents)
{
   ... // computes the average; cannot change score[]
}

...
```

  - **Returning an array**
    - Functions cannot return arrays, using a **`return`** statement.
    - However, an array can be returned, if it is embedded in a **`struct`**. (*see later*)
    - A function can return a pointer to an array (*see later*).

- **Multidimensional arrays**
  - An array to store the scores of the students for each exam question:

```
int score[NUMBER_STUDENTS][NUMBER_QUESTIONS];
```

  - Initialization of a multidimensional array when it is declared:
    - ```
      int m[2][3] = { {1,3,2} {5,2,9} };  // or ...
      int m[2][3] = { 1,3,2,5,2,9 };
      ```

  - Indexing a multidimensional array:
    - `score[0][0]` – score of the 1st student in the 1st question
    - `score[0][1]` – score of the 1st student in the 2nd question
    - ...
    - `score[1][2]` – score of the 2nd student in the 3rd question

    - NOTE: the "off-by-one offset" in the index ... :-(

  - **NOTE:**
    When a multidimensional array is used as a formal function parameter,
    the size of the first dimension is not given,
    but the remaining dimension sizes must be given in square brackets.
  - Since the first dimension size is not given, you usually need an additional parameter of type `int` that gives the size of this first dimension.

```
int readScores( int score[][NUMBER_QUESTIONS], int numStudents)
{
   ...
}
```
```

```
// 1D ARRAYS
// How they are stored in memory

#include <iostream>

using namespace std;

const int NMAX=3;

void main(void)
{
  int a[NMAX];
  int i;

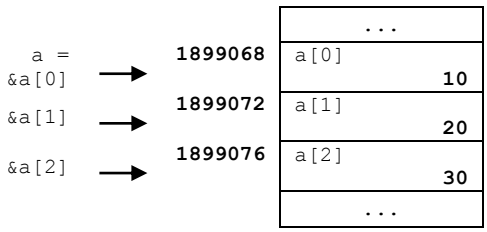
  for (i=0; i<NMAX; i++)
    a[i] = 10*(i+1);

  for (i=0; i<NMAX; i++)
    cout << "a[" << i << "] = " << a[i]
         << ",  &a[" << i << "] = " << (unsigned long) &a[i] << endl;

  cout << "a     = " << (unsigned long) a << endl;      // 'a' is ...
  cout << "&a[0] = " << (unsigned long) &a[0] << endl; // ... the address of a[0]

}
```

```
a[0] = 10,  &a[0] = 1899068
a[1] = 20,  &a[1] = 1899072
a[2] = 30,  &a[2] = 1899076
a     = 1899068
&a[0] = 1899068
```




TO DO BY STUDENTS:

- swap the contents of 2 arrays of the same size

```
// 2D ARRAYS
// How they are stored in memory
// How they are passed to function parameters

#include <iostream>

using namespace std;

const unsigned NLIN=2; // because NLIN & NCOL are globals, the dimensions of the array
const unsigned NCOL=3; // they can be omitted in the function parameters

int sumElems(const int m[NLIN][NCOL]) // NLIN could be ommited
{
  int sum=0;

  for (int i=0; i<NLIN; i++)
    for (int j=0; j<NCOL; j++)
      sum = sum + m[i][j];

  return sum;
}

//TO DO: function averageCols() - computes the average of the each of the columns of a[][]

void main(void)
{
  int a[NLIN][NCOL];

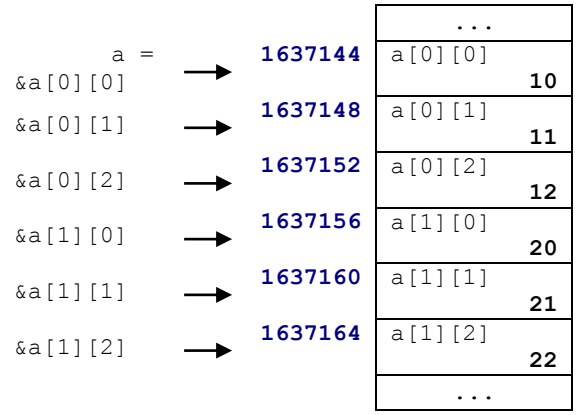
  for (int i=0; i<NLIN; i++)
    for (int j=0; j<NCOL; j++)
      a[i][j] = 10*(i+1)+j;

  for (int i=0; i<NLIN; i++)
    for (int j=0; j<NCOL; j++)
      cout << "a[" << i << "][" << j << "] = " << a[i][j]
           << ",  &a[" << i << "][" << j << "] = " << (unsigned long) &a[i][j]
           << endl;

  cout << "Sum of elements of a[][] = " << sumElems(a) << endl;
}
```

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 11 | 12 |
| 1 | 20 | 21 | 22 |

```
a[0][0] = 10,  &a[0][0] = 1637144
a[0][1] = 11,  &a[0][1] = 1637148
a[0][2] = 12,  &a[0][2] = 1637152
a[1][0] = 20,  &a[1][0] = 1637156
a[1][1] = 21,  &a[1][1] = 1637160
a[1][2] = 22,  &a[1][2] = 1637164
```



```
/*
ARRAYS
Passing arrays as function parameters
Counting number of occurrences of zero values in an array of integers
*/

#include <iostream>
#include <iomanip>

using namespace std;

void initArray(int v[], int size)
{

      for (int i = 0; i < size; i++)
      {
            v[i] = 10 * (i % 3); //try to guarantee the occurrence of some zeros
      }
}

void showArray(int v[], int size)
{
      for (int i = 0; i < size; i++)
      {
            cout << v[i] << endl;
      }
}

int countZeros(int v[], int size) // NOTE: arrays are passed "by reference"
{
      int numZeros=0;

      for (int i=0; i < size ; i++)
            if (v[i] = 0)  //BE CAREFUL !!!
                  numZeros++;

      return numZeros;
}

int main()
{
  const int MAX_SIZE = 10;
  int a[MAX_SIZE];

  int numElems;
  cout << " Effective number of elements (max = " << MAX_SIZE << ") ? ";
  cin >> numElems;

  initArray(a, numElems);
  showArray(a, numElems);
  cout << "number of zeros = " << countZeros(a, numElems) << endl;
  showArray(a, numElems); // NOTE: interpret the obtained result

  return 0;
}
```

```
/*
ARRAYS
Passing arrays as function parameters.
Using 'const' qualifier.
*/

#include <iostream>
#include <iomanip>

using namespace std;

void initArray(int v[], int size)
{

      for (int i = 0; i < size; i++)
      {
            v[i] = 10 * (i % 3);
      }
}

void showArray(const int v[], int size)
{
      for (int i = 0; i < size; i++)
      {
            cout << v[i] << endl;
      }
}

int countZeros(const int v[], int size)  //AFTER ADDING const ...
{
      int numZeros=0;

      for (int i=0; i < size ; i++)
            if (v[i] = 0)  //... THE COMPILER DETECTS THIS LOGIC ERROR
            // if (v[i] == 0) //CORRECT FORM ALTERNATIVE: if (0 == v[i])
                  numZeros++;  //              WHAT IS THE ADVANTAGE?

      return numZeros;
}

int main()
{
  const int MAX_SIZE = 10;
  int a[MAX_SIZE];

  int numElems;
  cout << "Effective number of elements (max = " << MAX_SIZE << ") ? ";
  cin >> numElems;

  initArray(a, numElems);
  showArray(a, numElems);
  cout << "number of zeros = " << countZeros(a, numElems) << endl;
  showArray(a, numElems);

  return 0;
}
```

TO DO BY STUDENTS:

- remove zeros from an array;
- compute mean and standard deviation of array elements;

```
/*
MULTIDIMENSIONAL ARRAYS - 2D ARRAYS
Reads quiz scores for each student into the two-dimensional array grade.
Computes the average score for each student and
the average score for each quiz.
Displays the quiz scores and the averages.
*/

#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

const int NUMBER_STUDENTS = 4, NUMBER_QUIZZES = 3;

void fill_grades(int grade[][NUMBER_QUIZZES]);

void compute_st_ave(const int grade[][NUMBER_QUIZZES], double st_ave[]);

void compute_quiz_ave(const int grade[][NUMBER_QUIZZES], double quiz_ave[]);

void display(const int grade[][NUMBER_QUIZZES], const double st_ave[], const
double quiz_ave[]);

//--------------------------------------------------------------------------

int main( )
{
      int grade[NUMBER_STUDENTS][NUMBER_QUIZZES];
      double st_ave[NUMBER_STUDENTS];
      double quiz_ave[NUMBER_QUIZZES];

      fill_grades(grade); // randomly !!!
      compute_st_ave(grade, st_ave);
      compute_quiz_ave(grade, quiz_ave);
      display(grade, st_ave, quiz_ave);
      return 0;
}
//--------------------------------------------------------------------------

void fill_grades(int grade[][NUMBER_QUIZZES]) // ..... fill... RANDOMLY !
{
      for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
            for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
                  grade[st_num-1][quiz_num-1] = 10 + rand() % 11;
}
//--------------------------------------------------------------------------

void compute_st_ave(const int grade[][NUMBER_QUIZZES], double st_ave[])
{
      for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
      {
            double sum = 0;
            for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
                  sum = sum + grade[st_num-1][quiz_num-1];

            st_ave[st_num -1] = sum/NUMBER_QUIZZES;

      }
}
//--------------------------------------------------------------------------
```

```
void compute_quiz_ave(const int grade[][NUMBER_QUIZZES], double quiz_ave[])
{
      for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
      {
            double sum = 0;
            for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
                  sum = sum + grade[st_num -1][quiz_num -1];

            quiz_ave[quiz_num -1] = sum/NUMBER_STUDENTS;

      }
}
//-------------------------------------------------------------------------

void display(const int grade[][NUMBER_QUIZZES], const double st_ave[],
const double quiz_ave[])
{
      cout.setf(ios::fixed);
      cout.setf(ios::showpoint);
      cout.precision(1);

      cout << setw(10) << "Student"
            << setw( 5) << "Ave"
            << setw(12) << "Quizzes\n";

      for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
      {
         cout << setw(10) << st_num << setw(5) << st_ave[st_num-1] << " ";
         for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
            cout << setw(5) << grade[st_num-1][quiz_num-1];
         cout << endl;
      }

      cout << "Quiz averages = ";
      for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
            cout << setw(5) << quiz_ave[quiz_num-1];
      cout << endl;
}
```

```
/*
2D ARRAYS -
ALTERNATIVE SOLUTION FOR THE PREVIOUS "PROBLEM" (i.e.index of 1st element is zero)
Reads quiz scores for each student into the two-dimensional array grade.
Computes the average score for each student and
the average score for each quiz.
Displays the quiz scores and the averages.
*/
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

const int NUMBER_STUDENTS = 4, NUMBER_QUIZZES = 3;

void fill_grades(int grade[][NUMBER_QUIZZES]);

void compute_st_ave(const int grade[][NUMBER_QUIZZES], double st_ave[]);

void compute_quiz_ave(const int grade[][NUMBER_QUIZZES], double
quiz_ave[]);

void display(const int grade[][NUMBER_QUIZZES], const double st_ave[],
const double quiz_ave[]);
//-------------------------------------------------------------------------

int main( )
{
      int grade[NUMBER_STUDENTS][NUMBER_QUIZZES];
      double st_ave[NUMBER_STUDENTS];
      double quiz_ave[NUMBER_QUIZZES];

      fill_grades(grade);
      compute_st_ave(grade, st_ave);
      compute_quiz_ave(grade, quiz_ave);
      display(grade, st_ave, quiz_ave);
      return 0;
}
//-------------------------------------------------------------------------

void fill_grades(int grade[][NUMBER_QUIZZES])
{
      for (int st_num = 0; st_num < NUMBER_STUDENTS; st_num++)
            for (int quiz_num = 0; quiz_num < NUMBER_QUIZZES; quiz_num++)
                  grade[st_num][quiz_num] = 10 + rand() % 11;
}
//-------------------------------------------------------------------------

void compute_st_ave(const int grade[][NUMBER_QUIZZES], double st_ave[])
{
      for (int st_num = 0; st_num < NUMBER_STUDENTS; st_num++)
      {
            double sum = 0;
            for (int quiz_num = 0; quiz_num < NUMBER_QUIZZES; quiz_num++)
                  sum = sum + grade[st_num][quiz_num];

            st_ave[st_num] = sum/NUMBER_QUIZZES;
      }
}
//-------------------------------------------------------------------------
```

```
void compute_quiz_ave(const int grade[][NUMBER_QUIZZES], double quiz_ave[])
{
      for (int quiz_num = 0; quiz_num < NUMBER_QUIZZES; quiz_num++)
      {
            double sum = 0;
            for (int st_num = 0; st_num < NUMBER_STUDENTS; st_num++)
                  sum = sum + grade[st_num][quiz_num];

            quiz_ave[quiz_num] = sum/NUMBER_STUDENTS;
      }
}
//-------------------------------------------------------------------------

void display(const int grade[][NUMBER_QUIZZES], const double st_ave[],
const double quiz_ave[])
{
      cout.setf(ios::fixed);
      cout.setf(ios::showpoint);
      cout.precision(1);
      cout << setw(10) << "Student"
            << setw(5) << "Ave"
            << setw(15) << "Quizzes\n";
      for (int st_num = 0; st_num < NUMBER_STUDENTS; st_num++)
      {
            cout << setw(10) << st_num + 1
                  << setw(5) << st_ave[st_num] << " ";
            for (int quiz_num = 0; quiz_num < NUMBER_QUIZZES; quiz_num++)
                  cout << setw(5) << grade[st_num][quiz_num];
            cout << endl;
      }

      cout << "Quiz averages =   ";
      for (int quiz_num = 0; quiz_num < NUMBER_QUIZZES; quiz_num++)
            cout << setw(5) << quiz_ave[quiz_num];
      cout << endl;
}
```

==========

## STRUCTS & typedef

==========

- **STRUCTs**
  - A structure is a **user-definable type**.
  - It is a derived data type,
    constructed using objects of other types (ex: int, string, ... or structures !).
  - The keyword `struct` introduces the structure definition:

```
struct Person // the new type is named "Person" – C++ syntax
{
  string name; // string type will be introduced later
  char gender;
  unsigned int age;
}; // COMMON ERROR: forgetting the semicolon

Person p1, p2; // p1 and p2 are variables of type Person
```

  - Suggestion: use an uppercase letter for the first character of the type name.
  - After you define the type, you can create variables of that type.
  - Thus, creating a structure is a two-part process.
    - First, you define a structure description that describes and labels the different types of data that can be stored in a structure.
    - Then, you can create structure variables, or, more generally, structure data objects, that follow the description's plan.

- **Accessing the fields of a structure**

```
cout << p1.name << "-" << p1.gender << ... ... << endl;
```

- **typedef**
  - The keyword `typedef` provides a mechanism for creating synonims (or aliases) for (previously defined) data types:

```
typedef unsigned int IdNumber;
IdNumber id;
```

    - creates type `IdNumber` that is the same as `unsigned int`
    - variable type of `id` is `IdNumber`

- Alternative way to create type `Person` using **typedef**:

```
typedef struct // the new type is named "Person"- C/C++ syntax
{
  string name;
  char gender;
  unsigned int age;
} Person;

Person p1 = {"Rui", 'M', 20}; //declaration w/initialization
```

- Using **typedef** to create another user defined type:

```
typedef unsigned int uint; // uint is the same as unsigned int
uint x;  // x is of type uint
```

```
=========================================================================
/*
- CREATING NEW TYPES
- USING STRUCTURES FOR RETURNING MULTIPLE VALUES FROM FUNCTIONS
*/

#include <iostream>
#include <string>

using namespace std;


struct Person
{
  string name;
  char gender;
  unsigned int age;
};


const unsigned NUM_MAX_PERSONS = 10;
// ☹ different n.o of persons => modify & recompile


Person readPerson()  // does not deal with invalid inputs
{
  Person p;
  cout << "Name ? ";  // ONLY ONE WORD ... I'll be back to this later
  cin >> p.name;
  cout << "Gender ? ";
  cin >> p.gender;
  cout << "Age ? ";
  cin >> p.age;

  return p;  // NOTE: a function can return a struct
}


int main()
{
  Person persons[NUM_MAX_PERSONS];
  size_t numPersons;

  // Read and validate number of persons
  cout << "How many persons ? ";
  cin >> numPersons;
  if (numPersons > NUM_MAX_PERSONS)
  {
    cerr << "Number of persons greater than allocated space ...!\n";
    exit(1);
  }

  // Read all person's data
  for (size_t i = 0; i < numPersons; i++)
  {
    persons[i] = readPerson();
  }
  // TO DO: show all the input data & process (ex:obtain name and gender of oldest person)
}
=========================================================================
```

```
=========================================================================
/*
- USING STRUCTURES FOR RETURNING ARRAYS FROM FUNCTIONS
- CREATING NEW TYPES
*/

#include <iostream>
#include <iomanip>

using namespace std;

const int SIZE = 10;

//typedef struct {int a[SIZE];} StructArr; // a new type is created; C-style
struct StructArr {int a[SIZE];}; // a new type is created; C++-style

StructArr initArray(int size)
{
      StructArr s;

      for (int i = 0; i < size; i++)
      {
            s.a[i] = 10 * (i % 3);
      }
      return s;
}

void showArray(const StructArr &s, int size)
{
      for (int i = 0; i < size; i++)
      {
            cout << s.a[i] << endl;
      }
}

int countZeros(const StructArr &s, int size)
{
      int numZeros=0;

      for (int i=0; i < size ; i++)
            if (s.a[i] == 0)
                  numZeros++;

      return numZeros;
}

int main()
{

      StructArr sa;

      sa = initArray(SIZE);
      showArray(sa,SIZE);

      cout << "number of zeros = " << countZeros(sa, SIZE) << endl;

      return 0;
}
=========================================================================
```

==========

## STL (Standard Template Library) VECTORS

==========

- **Vectors**
  - Vectors are a kind of **STL container**
  - Vectors are like arrays that can change size as your program runs :-)
  - Vectors, like arrays, have a base type
  - To declare an empty vector with base type `int`:

```
vector<int> v1;  // be careful! v1 is empty
```

    - `<int>` identifies **vector** as a **template class** (*see later*)

  - You can use any base type in a template class:

```
vector<double> v2(10); // v2 has 10 elements of type double
                       // all elements equal to zero
```

- **The vector library**
  - To use the vector class, include the vector library

```
#include <vector>
```

  - Vector names are placed in the standard namespace
    so the usual using directive is needed:

```
using namespace std;
```

- **Accessing vector elements**
  - Vectors elements are indexed in the range **[0.. vector_size-1]**
  - `[ ]`'s are used to read or change the value of an item:

```
for (size_t i = 0; i < v.size( ); i++)
   cout << v[i] << endl;
```

  - The member function `size()` returns the number of elements in a vector
  - The size of a vector is of type `size_t` (=> `#include <cstddef>`) ;
    `size_t` is an unsigned integer type
  - The member function `at()` can be used instead of operator `[ ]`
    to access the vector elements
    - `v[i]` – can be disastrous if `i` is out of the range **[0.. vector_size-1]**
    - `v.at(i)` - checks whether `i` is within the bounds,
      throwing an out_of_range exception if it is not
      (*see exception handling, later*)

- **Initializing vector elements**
  - `vector<int> v1;  // be careful! v1 is empty`
    - Elements are added to the end of a vector using the member function `push_back()`

```
v1.push_back(12);
v1.push_back(3);
v1.push_back(547);
```

  - `vector<int> v1;  // be careful! v1 is empty`
    `v1.resize(3);  // additional elements are set to zero`
    - after the above resizing,
      the vector has space for 3 elements
      so you can access `v[i]`, `i=0..2`
    - `resize()` can be also used to shrink a vector !
  - `vector<int> v2(10);  // v2 has 10 elements equal to 0`
    - elements of number types are initialized to zero
    - elements of other types are initialized using
      the default constructor of the class (*see later*)
    - `v2.size( )` would return `10`
  - `vector<int> v3(5,1);  // v3 has 5 elements equal to 1`
  - `vector<int> v4 = {10,20,30};  // possible with C++11 standard`
  - NOTE: it is also possible to initialize a vector from an array (*see later*)

- **Multidimensional vectors**
  - **Declaration**

```
//2D vector empty vector
vector< vector<int> > v1;

//2D vector with 5 lines and 3 columns
vector< vector<int> > v2(5, vector<int>(3));
```

  - **NOTE: in vectors, each row can have a different number of elements**
    **HOW TO DO THIS ?**

- **Accessing elements**

```
v2[3][1] = 10; // OR ...

v2.at(3).at(1) = 10;
```

- **Other vector methods**
  - see, for example: http://www.cplusplus.com/reference/vector/vector/
  - some of them will be introduced later

- **Vectors as function arguments / parameters and as return values**
  - Vector can be used as **call-by-value** or **call-by-reference parameters**
    - Large vectors that are not to be modified by the function should be passed as **const** call-by-reference parameters
  - **Functions can return vectors**
    - Large vectors that are to be modified by the function could be passed as call-by-reference parameters

```
==========================================================================
// VECTOR
// a kind of STL (Standard Template Library) container

#include <iostream>
#include <vector>
#include <cstddef> // where 'size_t' is defined

using namespace std;

/*
Returns all values within a range
Parameters:
 v - a vector of floating-point numbers
 low - the low end of the range
 high - the high end of the range
returns - a vector of values from v in the given range
*/
vector<double> between(vector<double> v, double low, double high)
// NOTE: vector v is passed by value
{
   vector<double> result;

   for (size_t i = 0; i < v.size(); i++)
      if (low <= v[i] && v[i] <= high)
         result.push_back(v[i]); //a vector can grow ...

   return result;  //a vector can be returned, unlike an array
}


int main()
{
   vector<double> salaries(5); //vector with 5 elements of type double
                               //try with vector<double> salaries;
   salaries[0] = 35000.0;
   salaries[1] = 63000.0;
   salaries[2] = 48000.0;
   salaries[3] = 78000.0;
   salaries[4] = 51500.0;

   vector<double> midrange_salaries = between(salaries, 45000.0, 65000.0);

   cout << "Midrange salaries:\n";
   for (size_t i = 0; i < midrange_salaries.size(); i++)
      cout << midrange_salaries[i] << "\n";

   return 0;
}
==========================================================================
```

```
================================================================================
/*
USING VECTORS
Read employee salaries
Determine those that have a mid-range salary
Raise their salary by a determined percentage
*/
#include <iostream>
#include <vector>
#include <cstddef>

using namespace std;

/*
Reads salaries
returns - a vector of salary values
*/
vector<double> readSalaries()
{
	int salary;
	vector<double> v;
	// ALTERNATIVES (to v.push_back()
	// 1) ask n.o of employees and do v.resize()
	// 2) declare vector only after asking the n.o of employees

	do
	{
		cout << "Salary (<=0 to terminate) ? ";
		cin >> salary;
		if (salary > 0)
			v.push_back(salary);
	} while (salary > 0);

	return v;
}


/*
Selects elements of vector v whose value belongs to the range [low..high].
Parameters:
 v - vector of values
 low - low end of the range
 high - high end of the range
*/
vector<double> vectorElemsBetween(vector<double> v, double low, double high)
{
	vector<double> result;

	for (size_t i = 0; i < v.size(); i++)
		if (low <= v[i] && v[i] <= high)
			result.push_back(v[i]);

	return result;
}
```

```
void showVector(vector<double> v)
{
      for (size_t i = 0; i < v.size(); i++)
            cout << v[i] << "\n";
}



/*
Raise all values in a vector by a given percentage.
Parameters:
 v - vector of values
 p - percentage to raise values by; values are raised p/100
*/
void raiseSalaries(vector<double> &v, double p)
{
      for (size_t i = 0; i < v.size(); i++)
            v[i] = v[i] * (1 + p / 100);
}



int main()
{
  const double MIDRANGE_LOW = 45000.0;
  const double MIDRANGE_HIGH = 65000.0;
  const double RAISE_PERCENTAGE = 1.0;

  vector<double> salaries;
  vector<double> midrangeSalaries;

  salaries = readSalaries();

  if (salaries.size() > 0)
  {
    cout << "Salaries between " << MIDRANGE_LOW << " and " << MIDRANGE_HIGH << ":\n";
    midrangeSalaries = vectorElemsBetween(salaries, MIDRANGE_LOW, MIDRANGE_HIGH);
    if (midrangeSalaries.size() > 0)
    {
      showVector(midrangeSalaries);
      raiseSalaries(midrangeSalaries,RAISE_PERCENTAGE);
      cout << "Raised salaries\n";
      showVector(midrangeSalaries);
    }
    else
      cout << "No salaries to be raised\n";
  }
  else
    cout << "No salaries to be processed\n";

  return 0;
}
```

==========

```
/*
USING VECTORS
Performance tip:
- for very large vectors, pass vectors to functions by reference;
  use qualifier const when vector can't be modified
Quality tip:
- using member function at() from vector class instead of operator []
  signals if the requested position is out of range

Program objective:
Read employee salaries
Determine those that have a mid-range salary
Raise their salary by a determined percentage
*/

#include <iostream>
#include <vector>
#include <cstddef>

using namespace std;

/*
Reads salaries
returns a vector containing the salaries
*/
vector<double> readSalaries()
{
	int salary;
	vector<double> v;

	do
	{
		cout << "Salary (<=0 to terminate) ? ";
		cin >> salary;
		if (salary > 0)
			v.push_back(salary);
	} while (salary > 0);

	return v;
}

/*
Selects elements of vector v whose value belongs to the range [low..high].
Parameters:
  v - vector of values
  low - low end of the range
  high - high end of the range
*/
vector<double> vectorElemsBetween(const vector<double> &v, double low,
double high)
{
	vector<double> result;

	for (size_t i = 0; i < v.size(); i++)
		if (low <= v.at(i) && v.at(i) <= high)
			result.push_back(v.at(i));

	return result;
}
```

```
/*
Shows a vector on screen, one value / line
*/
void showVector(const vector<double> &v)
{
	for (size_t i = 0; i < v.size(); i++)
		cout << v.at(i) << "\n";
}


/*
Raise all values in a vector by a given percentage.
Parameters:
  v - vector of values
  p - percentage to raise values by; values are raised p/100
*/
void raiseSalaries(vector<double> &v, double p)
{
	for (size_t i = 0; i < v.size(); i++)
		v.at(i) = v.at(i) * (1 + p / 100);
}


int main()
{
	const double MIDRANGE_LOW = 45000.0;
	const double MIDRANGE_HIGH = 65000.0;
	const double RAISE_PERCENTAGE = 10;

	vector<double> salaries;
	vector<double> midrangeSalaries;
	int numSalaries = 0;

	salaries = readSalaries();
	if (salaries.size() > 0)
	{
		cout << "Salaries between " << MIDRANGE_LOW << " and "
			<< MIDRANGE_HIGH << ":\n";
		midrangeSalaries =
			vectorElemsBetween(salaries, MIDRANGE_LOW, MIDRANGE_HIGH);
		if (midrangeSalaries.size() > 0)
		{
			showVector(midrangeSalaries);
			raiseSalaries(midrangeSalaries,RAISE_PERCENTAGE);
			cout << "Raised salaries\n";
			showVector(midrangeSalaries);
		}
		else
			cout << "No salaries to be raised\n";
	}
	else
		cout << "No salaries to be processed\n";

	return 0;
}
```

```
/*
VECTORS
The same program as "MULTIDIMENSIONAL ARRAYS - 2D ARRAYS" EXAMPLE,
using vectors instead of arrays

Reads quiz scores for each student into the two-dimensional array grade.
Computes the average score for each student and
the average score for each quiz.
Displays the quiz scores and the averages.
*/

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <vector>

using namespace std;

const int NUMBER_STUDENTS = 4, NUMBER_QUIZZES = 3;
// TO DO: USER CAN SPECIFY, IN RUNTIME, THOSE NUMBERS - SEE NEXT EXAMPLE

void fill_grades(vector< vector<int>> &grade);
// BE CAREFUL (in some compilers): NOT vector<vector<int>> . WHY ?
// Microsoft compiler does not care !

void compute_st_ave(const vector< vector<int> > &grade, vector<double> &st_ave);

void compute_quiz_ave(const vector< vector<int> > &grade, vector<double>
&quiz_ave);

void display(const vector< vector<int> > &grade, const vector<double> &st_ave,
             const vector<double> &quiz_ave);

//--------------------------------------------------------------------------

int main( )
{
      vector< vector<int> > grade(NUMBER_STUDENTS, vector<int>(NUMBER_QUIZZES));
      vector<double> st_ave(NUMBER_STUDENTS);
      vector<double> quiz_ave(NUMBER_QUIZZES);

      fill_grades(grade);
      compute_st_ave(grade, st_ave);
      compute_quiz_ave(grade, quiz_ave);
      display(grade, st_ave, quiz_ave);
      return 0;
}
//--------------------------------------------------------------------------


void fill_grades(vector< vector<int> > &grade)
{
      for (int st_num = 0; st_num < NUMBER_STUDENTS; st_num++)
            for (int quiz_num = 0; quiz_num < NUMBER_QUIZZES; quiz_num++)
                  grade[st_num][quiz_num] = 10; //10 + rand() % 11;
}
//--------------------------------------------------------------------------
```

```
void compute_st_ave(const vector< vector<int> > &grade, vector<double> &st_ave)
{
      for (int st_num = 0; st_num < NUMBER_STUDENTS; st_num++)
      {
            double sum = 0;
            for (int quiz_num = 0; quiz_num < NUMBER_QUIZZES; quiz_num++)
                  sum = sum + grade[st_num][quiz_num];

            st_ave[st_num] = sum/NUMBER_QUIZZES;
      }
}
//--------------------------------------------------------------------------


void compute_quiz_ave(const vector< vector<int> > &grade, vector<double> &quiz_ave)
{
      for (int quiz_num = 0; quiz_num < NUMBER_QUIZZES; quiz_num++)
      {
            double sum = 0;
            for (int st_num = 0; st_num < NUMBER_STUDENTS; st_num++)
                  sum = sum + grade[st_num][quiz_num];

            quiz_ave[quiz_num] = sum/NUMBER_STUDENTS;
      }
}
//--------------------------------------------------------------------------


void display(const vector< vector<int> > &grade, const vector<double> &st_ave,
             const vector<double> &quiz_ave)
{
      cout.setf(ios::fixed);
      cout.setf(ios::showpoint);
      cout.precision(1);

      cout << setw(10) << "Student"
            << setw(5) << "Ave"
            << setw(15) << "Quizzes\n";

      for (int st_num = 0; st_num < NUMBER_STUDENTS; st_num++)
      {
            cout << setw(10) << st_num + 1
                  << setw(5) << st_ave[st_num] << " ";
            for (int quiz_num = 0; quiz_num < NUMBER_QUIZZES; quiz_num++)
                  cout << setw(5) << grade[st_num][quiz_num];
            cout << endl;
      }

      cout << "Quiz averages =   ";
      for (int quiz_num = 0; quiz_num < NUMBER_QUIZZES; quiz_num++)
            cout << setw(5) << quiz_ave[quiz_num];
      cout << endl;
}
//--------------------------------------------------------------------------
```

```
/*
VECTORS - a BETTER solution: user chooses vector dimensions, in runtime
The same code as the previous one (using vectors instead of arrays)

Reads quiz scores for each student into the two-dimensional array grade.
Computes the average score for each student and
the average score for each quiz.
Displays the quiz scores and the averages.
*/

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <vector>
#include <cstddef>

using namespace std;

void fill_grades(vector< vector<int> > &grade);

void compute_st_ave(const vector< vector<int> > &grade, vector<double> &st_ave);

void compute_quiz_ave(const vector< vector<int> > &grade, vector<double> &quiz_ave);

void display(const vector< vector<int> > &grade, const vector<double> &st_ave, const
vector<double> &quiz_ave);


int main( )
{
      size_t numberStudents, numberQuizzes;

      cout << "Number of students ? "; cin >> numberStudents;
      cout << "Number of quizzes  ? "; cin >> numberQuizzes;

      vector< vector<int> > grade(numberStudents, vector<int> (numberQuizzes));
      vector<double> st_ave(numberStudents);
      vector<double> quiz_ave(numberQuizzes);

      fill_grades(grade);
      compute_st_ave(grade, st_ave);
      compute_quiz_ave(grade, quiz_ave);
      display(grade, st_ave, quiz_ave);
      return 0;
}

void fill_grades(vector< vector<int> > &grade)
{
      size_t numberStudents = grade.size();
      size_t numberQuizzes = grade[0].size();

      for (size_t st_num = 0; st_num < numberStudents; st_num++)
            for (size_t quiz_num = 0; quiz_num < numberQuizzes; quiz_num++)
                  grade[st_num][quiz_num] = 10 + rand() % 11;
}
```

```
void compute_st_ave(const vector< vector<int> > &grade, vector<double>
&st_ave)
{
      size_t numberStudents = grade.size();
      size_t numberQuizzes = grade[0].size();

      for (size_t st_num = 0; st_num < numberStudents; st_num++)
      {
            double sum = 0;
            for (size_t quiz_num = 0; quiz_num < numberQuizzes; quiz_num++)
                  sum = sum + grade[st_num][quiz_num];
            st_ave[st_num] = sum/numberQuizzes;
      }
}


void compute_quiz_ave(const vector< vector<int> > &grade, vector<double>
&quiz_ave)
{
      size_t numberStudents = grade.size();
      size_t numberQuizzes = grade[0].size();

      for (size_t quiz_num = 0; quiz_num < numberQuizzes; quiz_num++)
      {
            double sum = 0;
            for (size_t st_num = 0; st_num < numberStudents; st_num++)
                  sum = sum + grade[st_num][quiz_num];

            quiz_ave[quiz_num] = sum/numberStudents;
      }
}


void display(const vector< vector<int> > &grade, const vector<double>
&st_ave, const vector<double> &quiz_ave)
{

      size_t numberStudents = grade.size();
      size_t numberQuizzes = grade[0].size();

      cout.setf(ios::fixed);
      cout.setf(ios::showpoint);
      cout.precision(1);
      cout << setw(10) << "Student"
            << setw(5) << "Ave"
            << setw(15) << "Quizzes\n";
      for (size_t st_num = 0; st_num < numberStudents; st_num++)
      {
            cout << setw(10) << st_num + 1
                  << setw(5) << st_ave[st_num] << " ";
            for (size_t quiz_num = 0; quiz_num < numberQuizzes; quiz_num++)
                  cout << setw(5) << grade[st_num][quiz_num];
            cout << endl;
      }

      cout << "Quiz averages =   ";
      for (size_t quiz_num = 0; quiz_num < numberQuizzes; quiz_num++)
            cout << setw(5) << quiz_ave[quiz_num];
      cout << endl;
}
```

```
/*
VECTORS - Yet another solution, using push_back()

Reads quiz scores for each student into the two-dimensional array grade.
Computes the average score for each student and
the average score for each quiz.
Displays the quiz scores and the averages.
*/

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <vector>
#include <cstddef>

using namespace std;

void fill_grades(vector< vector<int> > &grade, size_t numberStudents,
size_t numberQuizzes);

void compute_st_ave(const vector< vector<int> > &grade, vector<double>
&st_ave, size_t numberStudents, size_t numberQuizzes);

void compute_quiz_ave(const vector< vector<int> > &grade, vector<double>
&quiz_ave, size_t numberStudents, size_t numberQuizzes);

void display(const vector< vector<int> > &grade, const vector<double>
&st_ave, const vector<double> &quiz_ave, size_t numberStudents, size_t
numberQuizzes);


int main( )
{
      vector< vector<int> > grade;// how many elements has 'grade' vector ?
      vector<double> st_ave;      // and 'st_ave' & 'quiz_ave' vectors ?
      vector<double> quiz_ave;

      size_t numberStudents, numberQuizzes;

      cout << "Number of students ? "; cin >> numberStudents;
      cout << "Number of quizzes  ? "; cin >> numberQuizzes;
      fill_grades(grade,numberStudents,numberQuizzes);
      compute_st_ave(grade, st_ave, numberStudents, numberQuizzes);
      compute_quiz_ave(grade, quiz_ave, numberStudents, numberQuizzes);
      display(grade, st_ave, quiz_ave, numberStudents, numberQuizzes);
      return 0;
}

void fill_grades(vector< vector<int> > &grade, size_t numberStudents,
size_t numberQuizzes)
{
      for (size_t st_num = 0; st_num < numberStudents; st_num++)
      {
            vector<int> studentGrade;
            for (size_t quiz_num = 0; quiz_num < numberQuizzes; quiz_num++)
                  studentGrade.push_back(10 + rand() % 11);
            grade.push_back(studentGrade);
      }
}
```

```
void compute_st_ave(const vector< vector<int> > &grade, vector<double>
&st_ave, size_t numberStudents, size_t numberQuizzes)
{
      // numberStudents = grade.size();    //alternative to parameters
      // numberQuizzes = grade[0].size(); //alternative to parameters

      for (size_t st_num = 0; st_num < numberStudents; st_num++)
      {
            double sum = 0;
            for (size_t quiz_num = 0; quiz_num < numberQuizzes; quiz_num++)
                  sum = sum + grade[st_num][quiz_num];

            st_ave.push_back(sum/numberQuizzes);
            //WHY NOT st_ave[st_num] = sum/numberQuizzes; ???
      }
}


void compute_quiz_ave(const vector< vector<int> > &grade, vector<double>
&quiz_ave, size_t numberStudents, size_t numberQuizzes)
{
      for (size_t quiz_num = 0; quiz_num < numberQuizzes; quiz_num++)
      {
            double sum = 0;
            for (size_t st_num = 0; st_num < numberStudents; st_num++)
                  sum = sum + grade[st_num][quiz_num];

            quiz_ave.push_back(sum/numberStudents);
      }
}


void display(const vector< vector<int> > &grade, const vector<double>
&st_ave, const vector<double> &quiz_ave, size_t numberStudents, size_t
numberQuizzes)
{
      cout.setf(ios::fixed);
      cout.setf(ios::showpoint);
      cout.precision(1);
      cout << setw(10) << "Student"
            << setw(5) << "Ave"
            << setw(15) << "Quizzes\n";
      for (size_t st_num = 0; st_num < numberStudents; st_num++)
      {
            cout << setw(10) << st_num + 1
                  << setw(5) << st_ave[st_num] << " ";
            for (size_t quiz_num = 0; quiz_num < numberQuizzes; quiz_num++)
                  cout << setw(5) << grade[st_num][quiz_num];
            cout << endl;
      }

      cout << "Quiz averages =   ";
      for (size_t quiz_num = 0; quiz_num < numberQuizzes; quiz_num++)
            cout << setw(5) << quiz_ave[quiz_num];
      cout << endl;
}
```

==========

## STRINGS

==========

- In computer programming,
  a **string** is traditionally a sequence of characters,
  - either as a literal constant or
  - as some kind of variable.
- The latter may allow its elements to be mutated and/or the length changed,
  or it may be constant (after creation).

### C-Strings & C++-strings

- **C-strings** are stored as arrays of characters
  =>
  ```
  #include <cstring>
  ```

- **C++ strings** are objects of the String class, part of the `std` namespace
  =>
  ```
  #include <string>
  using namespace std;
  ```

---

## C - STRINGS

---

### C-string declaration & representation

- To declare a C-string variable, declare an array of characters:
  - `char s[10];`
- C-strings use the null character `'\0'` (character with ASCII code zero) to end a string;
  the null character immediately follows the last character of the string
- **Be careful**, don't forget to allocate space for the ending null char:
  - `char stringName[MAXIMUM_STRING_SIZE + 1];`
  - `MAXIMUM_STRING_SIZE` is some value that you must define
  - `+ 1` reserves the additional character needed by `'\0'`
- Declaring a C-string as `char s[10]` creates space for only nine characters
  - the null character terminator requires one space

### Initializing a C-string

- Initialization of a C-string during declaration (bad solution):
  - `char salut[ ] = {'H','i','!','\0'}; // NOTE the ending '\0'`
- Better alternative:
  - `char salut[ ] = "Hi!";  // the null char '\0' is added for you`
- `char anotherSalut[20] = "Hi there!";`
  `// the characters with index  10..19 have an undetermined value`
- NOTE:
  - do not to replace the null character
    when manipulating indexed variables in a C-string
  - If the null character is "lost", the array cannot act like a C-string

### C-string Output

- C-strings can be written with the insertion operator (<<)
- Example:

```
char msg[ ] = "Hello";
cout << msg << " world!" << endl;
```

### C-string Input

- C-strings can be read with the extraction operator (>>)
- NOTE:
  - whitespace (`' ', \n, \t, ...`) ends reading of data ;
  - whitespace remains in the input buffer
- Example:

```
char name[80];
cout << "Your name?  " << endl;
cin >> name; // enter "Rui Sousa"; " Sousa" remains in the buffer!
```

### Reading an entire Line

- Predefined member function `getline()` can read an entire line, including spaces
- `getline()` is a **member function** of all input streams
  - `istream& getline (char* s, streamsize n );`
  - `istream& getline (char* s, streamsize n, char delim );`
- Calling: `streamName.getline(.....);`

- `cin.getline()`
  - extracts characters from the stream as unformatted input and
  - stores them into `s` as a **C-string**,
  - until either the extracted character is the delimiting character (`'\n'` or `delim`),
  - or `n` characters have been written to `s` (including the terminating null character); in this case, `getline()` stops even if the end of the line has not been reached.
- The delimiting character is:
  - the newline character (`'\n'`) for the first form of `getline()`, and
  - `delim` for the second form.
- When found in the input sequence, the delimiting character,
  - is extracted from the input sequence,
  - but discarded and not written to `s`.
- NOTE:
  - If the function stops reading because `n` characters have been read without finding the delimiting character, the failbit internal flag is set (=> `cin.clear()`) but the additional characters are removed from the buffer.

### Accessing string elements

- The elements of a string are accessed just like the elements of an array.
- Example:

```
char s[5]="Hi!";
s[1] = 'o';
cout << s << endl; // What is the output?
```

- Be careful, when accessing the elements of a string.
  - Do not access characters past the end of the array of chars!
  - When modifying it, do not forget that the ending null char must be present

### Assignment

- The assignment operator does not work with C-strings
- This statement is illegal:
  - `a_string = "Hello";`
  - this is an assignment statement, not an initialization.

- A common method to assign a value to a C-string variable is to use function **strcpy()**, defined in the cstring library
- Example:

```
char msg[10];
strcpy (msg, "Hello"); // places "Hello" followed by '\0' in  msg
```

- NOTE: **strcpy()** can create problems if not used carefully
  - **strcpy()** does not check the declared length of destination string
  - it is possible for **strcpy()** to write characters beyond the declared size of the array (*see* **strncpy()** )

### Comparison

- The == operator does not work as expected with C-strings.
- The predefined function **strcmp()** is used to compare C-string variables
  - `int strcmp ( const char str1[ ], const char str2[ ] );`
    *//why is the number of chars of each string not needed*
    *as in other functions that have parameters of type 'array'?*
  - As we shall see later, this prototype can also be written as:

```
int strcmp ( const char *str1, const char *str2 );
```

- **strcmp()** returns an integral value indicating the relationship between the strings:
  - a zero value indicates that both strings are equal;
  - a value greater than zero indicates that the first character that does not match has a greater value in **str1** than in **str2** ;
  - a value less than zero indicates the opposite.
- Example:

```
if (strcmp(cstr1, cstr2))
     cout << "Strings are not the same.";
else
     cout << "String are the same.";
```

### Converting C-strings to numbers

- "1234" and "12.3" are strings of characters
- 1234 and 12.3 are numbers

- There functions for converting strings to numbers (=> `#include <cstdlib>`)
  - `atoi()`– convert C-string to integer
  - `atol()` – convert C-string to long integer
  - `atof()` – convert C-string to double
- Example:

  `atoi("1234")` returns the integer 1234
  `atoi("#123")` returns 0 because `#` is not a digit

  `atof("9.99")` returns 9.99
  `atof("$9.99")` returns 0.0 because `$` is not a digit

### Concatenation

- `strcat()` concatenates two C-strings

### Other C-string operations

- *see table in the next pages*

### C-strings as arguments and parameters

- C-string variables are arrays
- C-string arguments and parameters are used just like arrays

---

## The standard string class (C++ strings)

---

### The standard string Class

- The **`string`** class allows the programmer to treat strings as a basic data type.
- **No need to deal with the implementation as with C-strings.**
- The string class is defined in the **string** library
  and the names are in the standard namespace
- To use the string class you need these lines:

```
#include <string>
using namespace std;
```

### Declaration and assignment of Strings

- The default string constructor initializes the string to the empty string
  - *class constructors will be introduced later*
- Another string constructor takes a C-string argument
- Example:

```
string phrase;        // empty string
string name("John"); // calls the string constructor
```

- **Variables of type string can be assigned with the = operator**
- Example:

```
string s1,s2,s3;
...
s1 = "Hello Mom!";
...
s3 = s2;
```

### I/O with class string

- The insertion operator << is used to output objects of type string
- Example:

```
string s = "Hello Mom!";
cout << s;
```

- The extraction operator >> can be used to input data for objects of type string
- Example:

```
string s1;
cout << "What is your name ? " ; cin >> s1;
```

- NOTE:
  - whitespace (`' ', \n, \t, ...`) ends reading of data ;
  - whitespace remains in buffer

### Accessing string elements

- characters in a string object can be accessed as if they are in an array
- as in an array, index values are not checked for validity!
- `at()` is an alternative to using `[ ]`'s to access characters in a string.
- `at()` checks for valid index values
- Example:

```
string str("Mary");
cout << str[6] << endl; // INVALID ACCESS ... DETECTED ...?
cout << str.at(6) << endl; // INVALID ACCESS IS DETECTED
```

### Comparison of strings

- **Comparison operators work with string objects.**
- Objects are compared using lexicographic order
  (alphabetical ordering using the order of symbols in the ASCII character set.)
- `==` returns true if two string objects contain the same characters in the same order
  - remember `strcmp()` for C-strings? :-(
- `<, >, <=, >=` can be used to compare string objects

### Strings concatenation

- Variables of type string can be concatenated with the `+ operator`
- Example:

```
s3 = s1 + s2;
```

  - If `s3` is not large enough to contain `s1 + s2`, more space is allocated

### String length

- The string class member function **`length`**
  returns the number of characters in the string object:
- Example:

```
size_t  n = s.length( );
```

### Converting C-strings to string objects

- Recall the automatic conversion from C-string to string:

```
char cstr[ ] = "C-string";
string str = cstr;
```

### Converting strings to C-strings

- The string class member function **`c_str()`**
  returns the C-string version of a string object
- Example:

```
strcpy(cstringVariable, stringVariable.c_str( ) );
```

### Mixing strings and C-strings

- It is natural to work with strings in the following manner:

```
string phrase = "I like " +  noun + "!";
```

- It is not so easy for C++!
  It must either convert the null-terminated C-strings, such as "I like",  to strings,
  or it must use an overloaded **operator+** that works with strings and C-strings

### getline for type 'string'

- A **`getline()`** function exists to read entire lines into a **string** variable
- This version of **`getline`** is not a member of the **`istream`** class,
  it is a non-member function.
- **`getline()`** declarations:
  - `istream& getline (istream &is, string &str, char delim);`
  - `istream& getline (istream &is, string &str);`
- Extracts characters from **`is`** (input stream) and stores them into **`str`** until
  - the delimitation character **`delim`** is found (1st prototype)
  - or the newline character, **`'\n'`** ( 2nd prototype)

- The extraction also stops
  - if the end of file is reached in `is` *(see later)*
  - or if some other error occurs during the input operation.
- If the delimiter is found,
  - it is extracted and discarded, i.e.
    it is not stored and the next input operation will begin after it.
- Example:

```
cout "Enter your full name:\n"; //now you can enter "Rui Sousa" ☺
getline(cin, name);
```

**Mixing "cin >>" and "getline" (BE CAREFUL !!!)**

- Recall `cin >>` skips whitespace to find what it is to read then stops reading when whitespace is found
- `cin >>` leaves the `'\n'` character in the input stream
- Example:

```
int n;
string line;
cin >> n; // leaves the '\n' in the input buffer
getline(cin, line); // returns immediately;
                    // 'line' is set equal to the empty string.
```

**Other string operations**

- *Ex: finding substrings => consult some manuals / web pages*

==========

## STRINGS

### ARRAY OF STRUCTS

==========

```
/*
C STRINGS
are arrays of characters, terminated by a null character, '\0'
*/

#include <iostream>
#include <iomanip>
#include <cstring>

using namespace std;

int main()
{
	const int MAX_NAME_LEN = 10;

	//string declarations C-style
	char name[MAX_NAME_LEN];
	char salutation[] = "Hello "; // string declaration with initialization

	cout << "Your name ? "; //try with "Rui" "Alexandrino" and "Rui Sousa"
	cin >> name;
	cout << salutation << name << "!\n";
	//cout << sizeof(salutation) << endl;

	/*
	//show 'name' characters (not only ...)
	for (unsigned i=0; i<MAX_NAME_LEN; i++) //TRY: for (unsigned i=0; i<strlen(name);
i++)
		cout << setw(4) << (unsigned) name[i] << " - " << name[i] << endl;
	*/

	return 0;
}
```

```
/*
C++ STRINGS
*/

#include <iostream>
#include <string>

using namespace std;

int main()
{

	string name; //string declaration; MAX. LENGTH NOT NECESSARY :-)

	cout << "Your name ? "; //try with "Rui" and "Rui Sousa"
	cin >> name;
	cout << "Hello " << name << "!\n";

	return 0;
}
--------------------------------------------------------------------------------
/*
C++ STRINGS
getline()
string member funcions call: length(), find_last_of(), substr()
*/

#include <iostream>
#include <string>
#include <cstddef>

using namespace std;

int main()
{

	string name, lastName;
	size_t posLastSpace;

	cout << "Your full name ? "; //try with "Rui" and "Rui Sousa"
	getline(cin,name);

	cout << "Hello " << name << "!\n";

	posLastSpace = name.find_last_of(' ');
	if (posLastSpace == string::npos) // no space character was found
		cout << "Your name has only one word ?!\n";
	else
	{
		lastName = name.substr(posLastSpace+1,name.length()-posLastSpace-1);
		cout << "Your last name is: " << lastName << endl;
	}

	return 0;
}

TO DO, BY THE STUDENTS:
search for other methods (similar to find_last_of(), substr() and length())
of the string class, for string manipulation.
The class and method concepts will be introduced later.
```

```cpp
/*
C++ STRINGS
string concatenation
*/

#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

int main()
{

	string name, salutation;

	cout << "Your name ? ";
	getline(cin,name);
	salutation = "Hello " + name + "!\n"; //can't do this with C-strings
	cout << salutation;

	return 0;
}
```

---

```cpp
/*
C++ STRINGS
accessing string elements
passing string parameters by reference
*/

#include <iostream>
#include <iomanip>
#include <string>
#include <cctype>  //toupper()
#include <cstddef> //size_t

using namespace std;

void strToUpper(string &str)  // NOTE : string reference. WHY ?
{
	for (size_t i=0; i<str.length(); i++)
		str[i] = toupper(str[i]);
	   // str[i] = toupper(str.at(i)); //checks for valid index, i
}

int main()
{

	string name, salutation;

	cout << "Your name ? ";
	getline(cin,name);
	strToUpper(name);
	salutation = "Hello " + name + "!\n";
	cout << salutation;

	return 0;
}
```

```
/*
C++ STRINGS
array of strings
*/

#include <iostream>
#include <iomanip>
#include <string>
#include <cctype>  //toupper()
#include <cstddef> //size_t

using namespace std;

void readNames(string names[], size_t n)
{
      for (size_t i=0; i < n ; i++)
      {
            cout << "Name [" << i << "] ? ";
            getline(cin,names[i]);
      }
}

//shows names in array names[] right-aligned
void showNames(const string names[], size_t n)
{
      size_t maxNameLen = 0;
      for (size_t i=0; i < n ; i++)
            if (names[i].length() > maxNameLen)
                  maxNameLen = names[i].length();

      for (size_t i=0; i < n ; i++)
            cout << setw(maxNameLen) << names[i] << endl;

}

int main()
{
      const unsigned NUM_NAMES = 5;

      string names[NUM_NAMES];

      readNames(names,NUM_NAMES);
      showNames(names,NUM_NAMES);

      return 0;
}
```

TO DO, BY THE STUDENTS:
Do the same using a vector instead of an array.

```
/*
C++ STRINGS
be careful when mixing "getline()" and "cin >> variable"
*/

#include <iostream>
#include <iomanip>
#include <string>
#include <cctype>  //toupper()
#include <cstddef> //size_t

using namespace std;

void readNames(string names[], unsigned ages[], size_t n)
{
      for (size_t i=0; i < n ; i++)
      {
            cout << "Name [" << i << "] ? ";
            getline(cin,names[i]);
            cout << "Age [" << i << "] ? ";
            cin >> ages[i];
            // BE CAREFUL WHEN MIXING getline()  AND  cin >> variable
            // WHICH IS THE SOLUTION ?
      }
}

//shows names in vector nms[] right-aligned
void showNames(string names[], unsigned ages[], size_t n)
{
      size_t maxNameLen = 0;
      for (size_t i=0; i < n ; i++)
            if (names[i].length() > maxNameLen)
                  maxNameLen = names[i].length();

      for (size_t i=0; i < n ; i++)
            cout << setw(maxNameLen) << names[i] <<
            setw(3) << ages[i] << endl;

}

int main()
{
      const unsigned NUM_NAMES = 5;

      string names[NUM_NAMES];
      unsigned ages[NUM_NAMES];

      readNames(names,ages,NUM_NAMES);
      showNames(names,ages,NUM_NAMES);

      return 0;
}
```

```
/*
STRUCTS
ARRAY OF STRUCTS
Using typedef keyword to form an alias for a type
*/
#include <iostream>
#include <iomanip>
#include <string>
#include <cctype>
#include <cstddef>

using namespace std;

typedef struct
{
      string name;
      unsigned age;
} NameAge;
// typedef works both in C and C++;
// ALTERNATIVE C++, only
// struct NameAge {...};
// parameter types remain the same

void readNames(NameAge namesAndAges[], size_t n)
{
      for (size_t i=0; i < n ; i++)
      {
            cout << "Name [" << i << "] ? ";
            getline(cin,namesAndAges[i].name);
            cout << "Age [" << i << "] ? ";
            cin >> namesAndAges[i].age;
            cin.ignore(1000,'\n'); // solves the "mixing problem"
      }
}

//shows names in vector namesAndAges[] right-aligned
void showNames(NameAge namesAndAges[], size_t n)
{
      size_t maxNameLen = 0;
      for (size_t i=0; i < n ; i++)
            if (namesAndAges[i].name.length() > maxNameLen)
                  maxNameLen = namesAndAges[i].name.length();

      for (size_t i=0; i < n ; i++)
            cout << setw(maxNameLen) << namesAndAges[i].name <<
            setw(3) << namesAndAges[i].age << endl;

}

int main()
{
      const unsigned NUM_NAMES = 5;
      NameAge namesAndAges[NUM_NAMES]; // TO DO: use vectors instead of arrays

      readNames(namesAndAges,NUM_NAMES);
      cout << endl;
      showNames(namesAndAges,NUM_NAMES);

      return 0;
}
```

**Some Predefined C-String Functions in `<cstring>` (*part 1 of 2*)**

| Function | Description | Cautions |
|---|---|---|
| `strcpy(`*Target_String_Var*`,` *Src_String*`)` | Copies the C-string value *Src_String* into the C-string variable *Target_String_Var*. | Does not check to make sure *Target_String_Var* is large enough to hold the value *Src_String*. |
| `strncpy(`*Target_String_Var*`,` *Src_String, Limit*`)` | The same as the two-argument `strcpy` except that at most *Limit* characters are copied. | If *Limit* is chosen carefully, this is safer than the two-argument version of `strcpy`. Not implemented in all versions of C++. |
| `strcat(`*Target_String_Var*`,` *Src_String*`)` | Concatenates the C-string value *Src_String* onto the end of the C string in the C-string variable *Target_String_Var*. | Does not check to see that *Target_String_Var* is large enough to hold the result of the concatenation. |
| `strncat(`*Target_String_Var*`,` *Src_String, Limit*`)` | The same as the two-argument `strcat` except that at most *Limit* characters are appended. | If *Limit* is chosen carefully, this is safer than the two-argument version of `strcat`. Not implemented in all versions of C++. |
| `strlen(`*Src_String*`)` | Returns an integer equal to the length of *Src_String*. (The null character, `'\0'`, is not counted in the length.) | |
| `strcmp(`*String_1*`,` *String_2*`)` | Returns 0 if *String_1* and *String_2* are the same. Returns a value < 0 if *String_1* is less than *String_2*. Returns a value > 0 if *String_1* is greater than *String_2* (that is, returns a nonzero value if *String_1* and *String_2* are different). The order is lexicographic. | If *String_1* equals *String_2*, this function returns 0, which converts to `false`. Note that this is the reverse of what you might expect it to return when the strings are equal. |
| `strncmp(`*String_1*`,` *String_2, Limit*`)` | The same as the two-argument `strcat` except that at most *Limit* characters are compared. | If *Limit* is chosen carefully, this is safer than the two-argument version of `strcmp`. Not implemented in all versions of C++. |

**Member Functions of the Standard Class `string`**

| Example | Remarks |
|---|---|
| **Constructors** | |
| `string str;` | Default constructor creates empty `string` object `str`. |
| `string str("sample");` | Creates a `string` object with data `"sample"`. |
| `string str(a_string);` | Creates a `string` object `str` that is a copy of `a_string`; `a_string` is an object of the class `string`. |
| **Element access** | |
| `str[i]` | Returns read/write reference to character in `str` at index `i`. Does not check for illegal index. |
| `str.at(i)` | Returns read/write reference to character in `str` at index `i`. Same as `str[i]`, but this version checks for illegal index. |
| `str.substr(position, length)` | Returns the substring of the calling object starting at `position` and having `length` characters. |
| **Assignment/modifiers** | |
| `str1 = str2;` | Initializes `str1` to `str2`'s data, |
| `str1 += str2;` | Character data of `str2` is concatenated to the end of `str1`. |
| `str.empty( )` | Returns *`true`* if `str` is an empty string; *`false`* otherwise. |
| `str1 + str2` | Returns a string that has `str2`'s data concatenated to the end of `str1`'s data. |
| `str.insert(pos, str2);` | Inserts `str2` into `str` beginning at position `pos`. |
| `str.remove(pos, length);` | Removes substring of size `length`, starting at position `pos`. |
| **Comparison** | |
| `str1 == str2  str1 != str2` | Compare for equality or inequality; returns a Boolean value. |
| `str1 < str2  str1 > str2`<br>`str1 <= str2  str1 >= str2` | Four comparisons. All are lexicographical comparisons. |
| **Finds** | |
| `str.find(str1)` | Returns index of the first occurrence of `str1` in `str`. |
| `str.find(str1, pos)` | Returns index of the first occurrence of string `str1` in `str`; the search starts at position `pos`. |
| `str.find_first_of(str1, pos)` | Returns the index of the first instance in `str` of any character in `str1`, starting the search at position `pos`. |
| `str.find_first_not_of (str1, pos)` | Returns the index of the first instance in `str` of any character not in `str1`, starting the search at position `pos`. |

```
/*
C-STRINGS and C++-STRINGS
Converting between each other
Comparing strings
Nobody would use two different types of strings to do this !!!
Just for illustrating string conversions
*/

#include <iostream>
#include <iomanip>
#include <cstring>
#include <string>

using namespace std;

int main()
{
      const int MAX_CODE_LEN = 80;

      char code1[MAX_CODE_LEN];
      string code2;

      cout << "Type your code : "; // ex: A1B2 C3B4 D5E6
      cin.getline(code1,MAX_CODE_LEN); // cin.getline( ) ONLY FOR C-strings
      cout << code1 << endl;

      cout << "Retype your code : ";
      getline(cin,code2); // getline( ) ONLY FOR C++-strings
      cout << code2 << endl;

      // CHECKING WHETHER THE 2 CODES ARE EQUAL ...

      // version 1 - convert both strings to C-style strings
      char code2aux[MAX_CODE_LEN];
      strcpy(code2aux, code2.c_str());
     cout << "test1: ";
     if (strcmp(code1,code2aux) == 0)
            cout << "codes are equal\n";
     else
            cout << "codes are different\n";

      // version 2 - convert both strings to C++-style strings
      string code1aux(code1); //OR: string code1aux = string(code1);
     cout << "test2: ";
      if (code2 == code1aux)
            cout << "codes are equal\n";
      else
            cout << "codes are different\n";

      return 0;
}



TO DO BY STUDENTS:
investigate the behaviour of cin.getline()
when more than MAX_CODE_LEN characters are inserted
```

```
/*
READING WITHOUT ECHO
Microsoft C/C++ compiler specific
*/

#include <iostream>
#include <iomanip>
#include <cstring>
#include <string>
#include <conio.h>  //needed for _getch();

using namespace std;

int main()
{
      const char ENTER = 13;
      char ch;
      string password;

      cout << "Password ? ";

      while (( ch = _getch()) != ENTER) // Microsoft C/C++ compiler specific

      {
            password = password + ch;
            cout << "*";
      }

      cout << endl << password << endl; // you shouldn't do that ... :-)

      return 0;
}
--------------------------------------------------------------------------
/*
READING WITHOUT ECHO
*/
#include <iostream>
#include <iomanip>
#include <cstring>
#include <string>
#include <conio.h>

using namespace std;

int main()
{
      const char ENTER = 13;
      char ch;

      cout << "chars (end with Z) ? \n";
      do
      {
            ch = _getch(); //note: returns 13 (CARRIAGE RETURN) when ENTER key is typed
                           // TRY: cin.get()
            cout << ch << "-" << unsigned(ch) << endl;
      } while (ch != 'Z');

      return 0;
}
```