

1. The Context of Software Development
2. Writing a C++ Program
3. Values and Variables
4. Expressions and Arithmetic
5. Conditional Execution
6. Iteration
7. Other Conditional and Iterative Statements
- 8. Using Functions**
- 9. Writing Functions**
- 10. Managing Functions and Data**

11. Sequences
12. Sorting and Searching
13. Standard C++ Classes
14. Custom Objects
15. Fine Tuning Objects
16. Building some Useful Classes
17. Inheritance and Polymorphism
18. Memory Management
19. Generic Programming
20. The Standard Template Library
21. Associative Containers
22. Handling Exceptions

FUNCTIONS

Functions

Consider the following programs:

```
//we wish to evaluate 3 to the index 4
#include <iostream>
using namespace std;
int main() {
    int threeExpFour = 1;
    for (int i = 0; i < 4; i = i + 1) {
        threeExpFour = threeExpFour * 3;
    }
    cout << "3^4 is " << threeExpFour << endl;
    return 0;
}
```



```
//we wish to evaluate 3 to the index 4
#include <iostream>
using namespace std;
int main() {
    int threeExpFour = 1;
    for (int i = 0; i < 4; i = i + 1) {
        threeExpFour = threeExpFour * 3;
    }
    cout << "3^4 is " << threeExpFour << endl;

    //Now, we wish to evaluate 6 to the index 5,
    //So, we do a copy-paste of the previous program and we adapt it
    int sixExpFive = 1;
    for (int i = 0; i < 5; i = i + 1) {
        sixExpFive = sixExpFive * 6;
    }
    cout << "6^5 is " << sixExpFive << endl;
    return 0;
}
```

Functions

```
#include <iostream>
using namespace std;
int main() {
    int threeExpFour = 1; //we wish to evaluate 3 to the index 4
    for (int i = 0; i < 4; i = i + 1) {
        threeExpFour = threeExpFour * 3;
    }
    cout << "3^4 is " << threeExpFour << endl; //Now, we wish to evaluate 6 to the index 5,
    //So, we do a copy-paste of the previous program and we adapt it
    int sixExpFive = 1;
    for (int i = 0; i < 5; i = i + 1) {
        sixExpFive = sixExpFive * 6;
    }
    cout << "6^5 is " << sixExpFive << endl; //Now, we wish to evaluate 12 to the index 10,
    //So, we do a copy-paste of the previous program and we adapt it
    int twelveExpTen = 1;
    for (int i = 0; i < 10; i = i + 1) {
        twelveExpTen = twelveExpTen * 12;
    }
    cout << "12^10 is " << twelveExpTen << endl;
    return 0;
}
```

Functions

- The copy-paste strategy we used in the previous slides is “bad”.
- A function is a group of statements that is executed when it is **called** from some point of the program.

Hence, it can be used when we wish to re-use a code.

- The following is its format: `type name (parameter1, parameter2, ...) { statements }`

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

int raisepower(int a, int b){
    int z=1;
    for (int i=0; i<b; i++){
        z=a*z;
    }
    return (z);
}

int main() {
    int threeExpFour = raisepower(3, 4);
    cout << "3^4 is " << threeExpFour << endl;
    int sixExpFive = raisepower(6, 5);
    cout << "6^5 is " << sixExpFive << endl;
    int twelveExpTen = raisepower(12, 10);
    cout << "12^10 is " << twelveExpTen << endl;
    return 0;}
```

The parameters are sometimes called arguments

Functions

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

int raisepower(int a, int b){
    int z=1;
    for (int i=0; i<b; i++){
        z=a*z;
    }
    return (z);
}

int main() {
    int x,y;
    cout<<"enter the base number\n";
    cin>>x;
    cout<<"enter the exponent\n";
    cin>>y;
    int expo=raisepower( x, y);
    cout<<x <<" to the power "<< y<< " is "<< expo<<endl;
    return 0;}
```

Functions

- REMARK:

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

int main() {
    int x,y;
    cout<<"enter the base number\n";
    cin>>x;
    cout<<"enter the exponent\n";
    cin>>y;
    int expo=raisepower( x, y);
    cout<<x <<" to the power "<< y<< " is "<< expo<<endl;
    return 0;}

int raisepower(int a, int b){
    int z=1;
    for (int i=0; i<b; i++){
        z=a*z;
    }
    return (z);
}
```

Generally, compilers run programs from the top to the bottom. Hence, when it meets a variable, function, etc that was not initialized earlier, though it may be initialized later in your code, the compiler shall signal an error.

Functions

- REMARK:

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

int raisepower(int a, int b); // this method is known as PROTOTYPING a function

int main() {
    int x,y;
    cout<<"enter the base number\n";
    cin>>x;
    cout<<"enter the exponent\n";
    cin>>y;
    int expo=raisepower( x, y);
    cout<<x <<" to the power "<< y<< " is "<< expo<<endl;
    return 0;}

int raisepower(int a, int b){
    int z=1;
    for (int i=0; i<b; i++){
        z=a*z;
    }
    return (z);
}
```

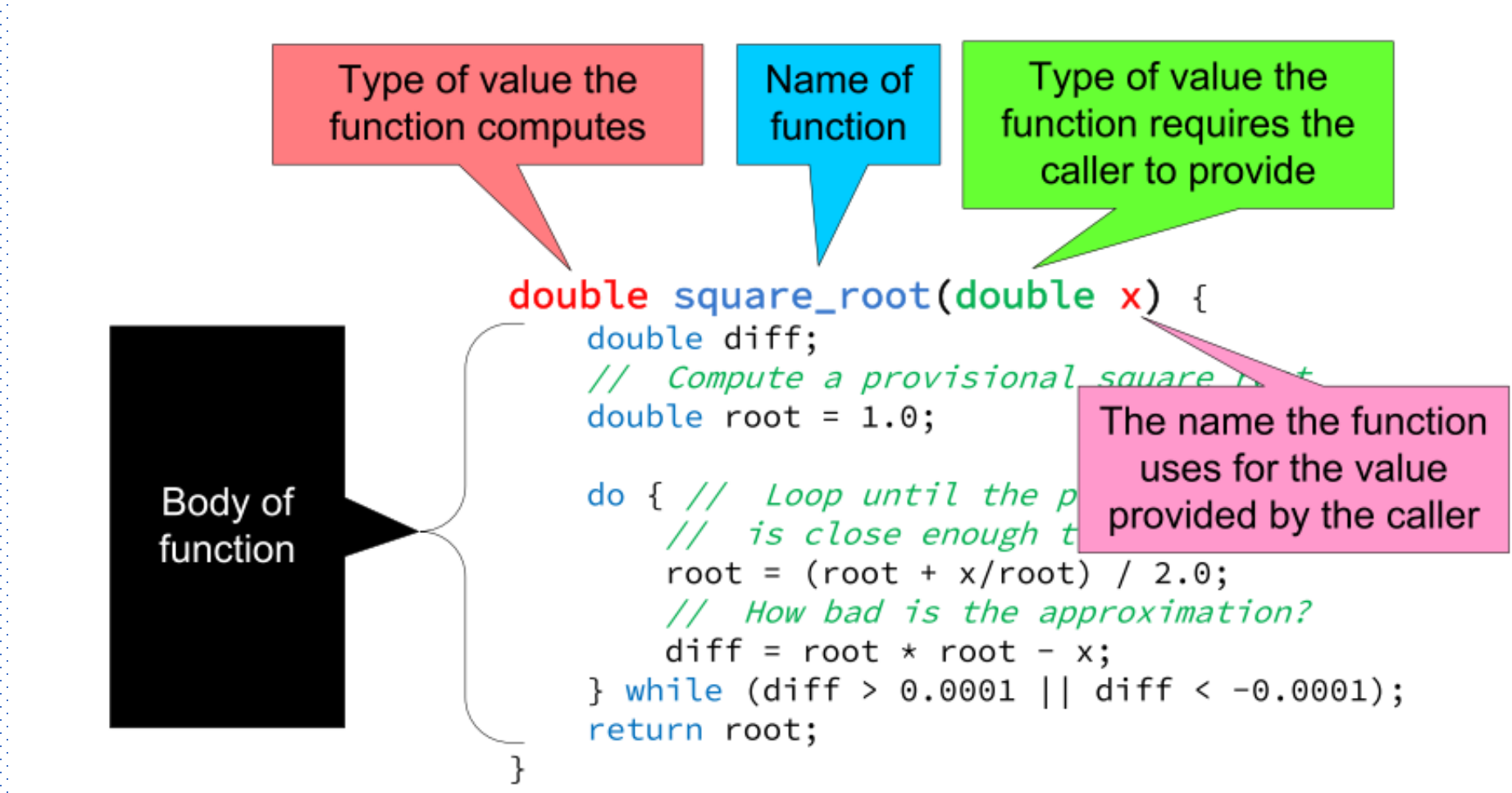
Argument order matters:

– *raisepower(2,3) is $2^3=8$*

– *raisepower(3,2) is $3^2=9$*

*One way to solve the error is to either describe the function before the main function as we did earlier or is to do as we did now (using a **prototype**). i.e. you inform the compiler you'll implement it later*

Functions



Functions

- Up to one value may be returned; it must be the same type as the return type.

```
int foo()
{
    return "hello"; // error
}
```

```
char* foo()
{
    return "hello"; // ok
}
```

- If no values are returned, give the function a void return type

```
void printNumber(int num) {
    cout << "number is " << num << endl;
}
```

```
int main() {
    printNumber(4); // number is 4
    return 0;
}
```

Note that you cannot declare a variable of type void

```
int main() {
    void x; // ERROR
    return 0;
}
```

Functions

Although it is optional to specify void in the parameter list. In C++, a parameter list can simply be left blank if we want a function with no parameters.

```
// void function example
#include <iostream>
using namespace std;

void printmessage ()
{
    cout << "I'm a function!";
}

int main ()
{
    printmessage ();
    return 0;
}
```

```
void printmessage (void)
{
    cout << "I'm a function!";
}
```

Functions

- Return statements don't necessarily need to be at the end.
- Function returns as soon as a return statement is executed.

```
void printNumberIfEven(int num) {  
    if (num % 2 == 1) {  
        cout << "odd number" << endl;  
        return;  
    }  
    cout << "even number; number is " << num << endl;  
}  
  
int main() {  
    int x = 4;  
    printNumberIfEven(x);  
    // even number; number is 3  
    int y = 5;  
    printNumberIfEven(y);  
    // odd number  
}
```

Argument Type Matters:

```
#include <iostream>  
using namespace std;  
  
void printnumber(int x)  
{  
    cout << x << " ";  
}  
  
void printcharacter(char *x)  
{  
    cout << x << endl;  
}  
  
int main() {  
    printnumber(4); printcharacter("courses");  
    return 0;}
```

Function overloading

Many functions with the same name, but different arguments. The function called is the one whose arguments match the invocation

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

void printnumber(int x)
{
    cout << x;
}

void printnumber(char *x)
{
    cout << x ;
}

void printnumber(int x, int y){
    cout << x*y << endl;
}

int main() {
    printnumber(4); printnumber(" or ");printnumber(2,3);
    return 0;}

```

4 or 6

Process returned 0 (0x0) execution time : 0.018 s
Press any key to continue.

Arguments passed by value and by reference.

Until now, in all the functions we have seen, the arguments passed to the functions have been passed *by value*. This means that when calling a function with parameters, what we have passed to the function were copies of their values but never the variables themselves.

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

// pass-by-value
void increment(int a) {
    a = a + 1;
    cout << "a in increment " << a << endl;
}

int main() {
    int q = 3;
    increment(q);
    cout << "q in main " << q << endl;
    return 0;}
```

```
a in increment 4
q in main 3
```

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

// pass-by-reference
void increment(int &a) {
    a = a + 1;
    cout << "a in increment " << a << endl;
}

int main() {
    int q = 3;
    increment(q);
    cout << "q in main " << q << endl;
    return 0;}
```

```
a in increment 4
q in main 4
```

Arguments passed by value and by reference.

The return statement only allows you to return 1 value. Passing output variables by reference overcomes this limitation.

```
// more than one returning value
#include <iostream>
using namespace std;
void prevnext (int x, int& prev, int& next)
{
    prev = x-1;
    next = x+1;
}
int main ()
{
    int x=100, y, z;
    prevnext (x, y, z);
    cout << "Previous=" << y << ", Next=" << z;
    return 0;}
```

Previous=99, Next=101

```
// more than one returning value, FALSE
#include <iostream>
using namespace std;
void prevnext (int x, int prev, int next)
{
    prev = x-1;
    next = x+1;
}
int main ()
{
    int x=100, y, z;
    prevnext (x, y, z);
    cout << "Previous=" << y << ", Next=" << z;
    return 0;}
```

Previous=7012244, Next=4309680

Arguments passed by value and by reference.

```
// more than one returning value
#include <iostream>
using namespace std;

int divide(int numerator, int denominator, int &remainder) {
    remainder = numerator % denominator;
    return numerator / denominator;
}

int main() {
    int num = 14;
    int den = 4;
    int rem;
    int result = divide(num, den, rem);
    cout << result << "*" << den << "+" << rem << "=" << num << endl;
    // 3*4+2=12
}
```

3*4+2=14

Functions: Use of cmath

You don't actually need to implement `raisepower` yourself; **cmath** (part of the standard library) contains functions **pow** and **sqrt**.

```
#include <iostream>
#include <cmath>
using namespace std;

double fourthRoot(double num) {
    return sqrt(sqrt(num));
}

int main () {

    cout<< fourthRoot(16);
    return 0;}
```

The `clock` function from the `<ctime>` library requests from the operating system the amount of time an executing program has been running.

```
#include <iostream>
#include <ctime>

int main() {
    char letter;
    std::cout << "Enter a character:";
    clock_t seconds = clock(); //Record starting time
    std::cin >> letter;
    clock_t other = clock(); //Record ending time
    std::cout << static_cast<double>(other - seconds) / CLOCKS_PER_SEC << "seconds\n";
}
```

Functions: Use of cmath

mathfunctions Module	
<code>double sqrt(double x)</code>	Computes the square root of a number: $\text{sqrt}(x) = \sqrt{x}$
<code>double exp(double x)</code>	Computes e raised a power: $\text{exp}(x) = e^x$
<code>double log(double x)</code>	Computes the natural logarithm of a number: $\text{log}(x) = \log_e x = \ln x$
<code>double log10(double x)</code>	Computes the common logarithm of a number: $\text{log}(x) = \log_{10} x$
<code>double cos(double)</code>	Computes the cosine of a value specified in radians: $\text{cos}(x) = \cos x$; other trigonometric functions include sine, tangent, arc cosine, arc sine, arc tangent, hyperbolic cosine, hyperbolic sine, and hyperbolic tangent
<code>double pow(double x, double y)</code>	Raises one number to a power of another: $\text{pow}(x, y) = x^y$
<code>double fabs(double x)</code>	Computes the absolute value of a number: $\text{fabs}(x) = x $

Others include **tan()** to evaluate the tangent, **floor()** to round down, **ceil()** to round up

Functions: Use of ctype

charfunctions Module	
<code>int toupper(int ch)</code>	Returns the uppercase version of the given character; returns the original character if no uppercase version exists (such as for punctuation or digits)
<code>int tolower(int ch)</code>	Returns the lowercase version of the given character; returns the original character if no lowercase version exists (such as for punctuation or digits)
<code>int isupper(int ch)</code>	Returns a nonzero value (true) if <code>ch</code> is an uppercase letter ('A' – 'Z'); otherwise, it returns 0 (false)
<code>int islower(int ch)</code>	Returns a nonzero value (true) if <code>ch</code> is a lowercase letter ('a' – 'z'); otherwise, it returns 0 (false)
<code>int isalpha(int ch)</code>	Returns a nonzero value (true) if <code>ch</code> is a letter from the alphabet ('A' – 'Z' or 'a' – 'z'); otherwise, it returns 0 (false)
<code>int isdigit(int ch)</code>	Returns a nonzero value (true) if <code>ch</code> is a digit ('0' – '9'); otherwise, it returns 0 (false)

Functions: Random numbers

```
# include <iostream >
# include <cstdlib > // C standard library -
                        // defines rand () , srand () , RAND_MAX
# include <ctime > // C time functions - defines time ()
using namespace std;

int main () {

    srand ( time (0) ); // Set the seed ;

                        // time (0) returns current time as a number

    int randNum = rand ();

    cout << "A random number : " << randNum << endl ;

    return 0;}
```

The C++ standard libraries include the **rand()** function for generating random numbers between 0 and RAND MAX (an integer constant defined by the compiler). These numbers are not truly random; they are a random-seeming but deterministic sequence based on a particular “seed” number. To make sure we don’t keep getting the same random-number sequence, we generally use the **current time as the seed number**.

Functions: Using many files

- We said earlier that our target when we create functions is to re-use them later. But so far, we couldn't really exploit this option since our functions were located in the `main()` function.
- **C++ allows us to divide our program in many source files. Each file contains one or several functions.** Hence, this gives us the possibility to include these files (thereby the functions that we need in our different projects).
- To make things clean, we need **two files** and not one:
 - One source file with extension **.cpp**: **contains the source code of the function;**
 - A header file with extension **.h**: **just contains the function prototype**

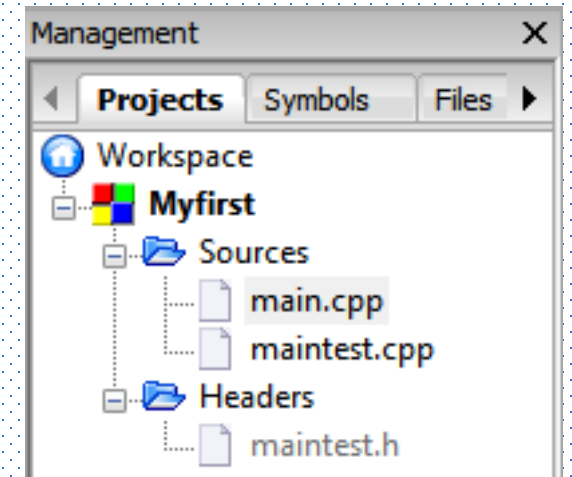
Functions: Using many files

Creating the .cpp file

- File > New > File.
- Select C/C++ source, then click on Go
- Choose a path (the path of your main.cpp) and write a name (the name should reflect the content for readability purposes).
- Select all the available options and click on finish

Creating the .h file

- File > New > File.
- Select C/C++ header, then click on Go
- Choose a path (the path of your main.cpp) and write a name (the name should be the name of the corresponding .cpp file created earlier but with extension .h for readability purposes).
- Select all the available options and click on finish



Functions: Using many files

After creating these files, we may now decide to use them as described earlier

```
main.cpp x maintest.cpp x maintest.h x
1  #include <iostream>
2  #include "maintest.h" // we always include the .h and not the .cpp
3  using namespace std;
4  int main()
5  {
6      int a(2), b(2);
7      cout << "Value of a : " << a << endl;
8      cout << "Value of b : " << b << endl;
9      b = addTwo(a); //calls the function
10     cout << "Value of a : " << a << endl;
11     cout << "Value of b : " << b << endl;
12     return 0;
13 }
```

```
main.cpp x maintest.cpp x maintest.h x
1  #include "maintest.h"
2  int addTwo(int numReciev)
3  {
4      int value(numReciev + 2);
5      return value;
6  }
```

```
main.cpp x maintest.cpp x *maintest.h x
1  #ifndef MAINTEST_H_INCLUDED
2  #define MAINTEST_H_INCLUDED
3
4  int addTwo(int numReciev); //prototype, don't forget the ;
5
6  #endif // MAINTEST_H_INCLUDED
7
8  /*write your codes between the last two default green codes;
9  they only appear by default when you use code blocks, else
10 you'll have to write them yourself; never modify those default
11 codes, they prevent the compiler from including the file multiple times*/
```

```
Value of a : 2
Value of b : 2
Value of a : 2
Value of b : 4
```

Functions: Using many files

By so doing, we can put several functions per file after grouping them into categories e.g. mathematic functions in one file, functions to display a menu in another file, functions to displace a game character in a third file.

Default values in parameters.

```
// default values in functions
#include <iostream>
using namespace std;
int divide (int a, int b=2)
{
    int r;
    r=a/b;
    return (r);
}

int main ()
{
    cout << divide (12);
    cout << endl;
    cout << divide (20,4);
    return 0;
}
```

6
5

```
#include <iostream>
using namespace std;
// Prototype with default values
int numberOfSecs(int hours, int minutes = 0, int seconds = 0);
// Main
int main()
{
    cout << numberOfSecs(1, 10, 25) << endl;
    return 0;
}

// Defining the function WITHOUT default values
int numberOfSecs(int hours, int minutes, int seconds)
{
    int total = 0;
    total = hours * 60 * 60;
    total += minutes * 60;
    total += seconds;
    return total;
}
```

When using the prototype method, the default values are specified in the prototype and NOT in the function definition!

If your code is divided into files, then specify it only in the header file .h

Default values in parameters.

```
#include <iostream>
using namespace std;
// Prototype with default values
int numberOfSecs(int hours, int minutes = 0, int seconds = 0);
// Main
int main()
{
    cout << numberOfSecs(1, 10, 25) << endl;
    return 0;
}
// Defining the function WITHOUT default values
int numberOfSecs(int hours, int minutes, int seconds)
{
    int total = 0;
    total = hours * 60 * 60;
    total += minutes * 60;
    total += seconds;
    return total;
}
```

When using the prototype method, the default values are specified in the prototype and NOT in the function definition!

If your code is divided into files, then specify it only in the header file .h

`cout << numberOfSecs(1,,25) << endl;` ❌

`cout << numberOfSecs(1,0,25) << endl;` ✅

`int numberOfSecs(int hours = 0, int minutes, int seconds);` ❌

//Error, default parameters must be on the right.

`int numberOfSecs(int seconds, int minutes, int hours = 0);` ✅

//OK

Exercises

1. Write a program that makes the output :

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
```

2. Write a program that displays a square filled with . and whose borders are made of x and whose size is given by the user. For example if the user enters 5, he will obtain :

```
XXXXX
X...X
X...X
X...X
XXXXX
```

Exercises

- 3. Write a program that display the 100 first terms of the Fibonacci sequence. We recall that $\text{fib}(0)=\text{fib}(1)=1$; $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$.*
- 4. Write a program that estimates PI by counting the number of points of a square which are in a given disc.*
- 5. Write a program that can swap two numbers inserted by the user. e.g. when he inserts 3 then 5, your program should return 5 and 3.*

ARRAYS AND STRINGS

Arrays

- We generally wish in most cases to have several variables of the same type that play nearly the same role. e.g. the list of users for a website: that may require a huge amount of string variables; or the 10 best scores of a game, etc.
- C++ just as many other languages propose a simple way to group these identical data in a single packet. This involves the use of **arrays**.
- In this section, we shall learn how to handle two types of arrays: the **static** (the size is known in advance, e.g. as that of the 10 best scores) and the **dynamic** (the size can vary, e.g. as the website example above).
- The size of the array is referred to as its *dimension*.

Arrays: static arrays

- We can observe that just to display the 5 best scores, we used many lines of codes. What if we wanted to display the best 100 scores?

```
int main()
{
    string nameBestPlayer1("Chris Prince");
    string nameBestPlayer2("Daniel Joseph");
    string nameBestPlayer3("Elume Raymond");
    string nameBestPlayer4("Koffi Kenneth");
    string nameBestPlayer5("Ojong Lovert");
    int bestScore1(118218);
    int bestScore2(100432);
    int bestScore3(87347);
    int bestScore4(64523);
    int bestScore5(31415);

    cout << "1) " << nameBestPlayer1 << " " << bestScore1 << endl;
    cout << "2) " << nameBestPlayer2 << " " << bestScore2 << endl;
    cout << "3) " << nameBestPlayer3 << " " << bestScore3 << endl;
    cout << "4) " << nameBestPlayer4 << " " << bestScore4 << endl;
    cout << "5) " << nameBestPlayer5 << " " << bestScore5 << endl;
    return 0;
}
```

```
1) Chris Prince 118218
2) Daniel Joseph 100432
3) Elume Raymond 87347
4) Koffi Kenneth 64523
5) Ojong Lovert 31415
```

Arrays: static arrays

- To declare an array in C++, we write the following:

type *arrayName*[*dimension*];

- The elements of an array can be accessed by using an *index* into the array. Arrays in C++ are zero-indexed, so the first element has an index of 0.
- Like normal variables, the elements of an array must be initialized before they can be used; otherwise we will almost certainly get unexpected results in our program.

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

int main() {

    int arr[4];
    cout << "Please enter 4 integers:" << endl;

    for(int i = 0; i < 4; i++)
        cin >> arr[i];
    cout << "Values in array are now:";

    for(int i = 0; i < 4; i++)
        cout << " " << arr[i];

    cout << endl;

    return 0;}

```

```
//arrays example
#include <iostream>
using namespace std;
int billy [] = {1, 2, 3, 4, 5};
int n, result=0;
int main ()
{
    for ( n=0 ; n<5 ; n++ )
    {
        result += billy[n];
    }
    cout << result;
    return 0;
}

```


Arrays: static arrays

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

```
int sum(const int array[], const int length) {
    long sum = 0;
    for(int i = 0; i < length; sum += array[i++]);
    return sum;
}
```

Size of the array

*Indicates that the
elements of the
array are constants*

```
int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7};
    cout << "Sum: " << sum(arr, 7) << endl;
    return 0;
}
```

```
// arrays as parameters
#include <iostream>
using namespace std;

void printarray (int arg[], int length) {
    for (int n=0; n<length; n++)
        cout << arg[n] << " ";
    cout << "\n";
}

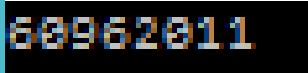
int main ()
{
    int firstarray[] = {5, 10, 15};
    int secondarray[] = {2, 4, 6, 8, 10};
    printarray (firstarray,3);
    printarray (secondarray,5);
    return 0;
}
```

```
5 10 15
2 4 6 8 10
```

It is important to note that arrays are *passed by reference* and so any changes made to the array within the function will be observed in the calling scope.

Arrays

```
int main() {  
    int twoDimArray[2][4];  
    twoDimArray[0][0] = 6;  
    twoDimArray[0][1] = 0;  
    twoDimArray[0][2] = 9;  
    twoDimArray[0][3] = 6;  
    twoDimArray[1][0] = 2;  
    twoDimArray[1][1] = 0;  
    twoDimArray[1][2] = 1;  
    twoDimArray[1][3] = 1;  
  
    for(int i = 0; i < 2; i++)  
        for(int j = 0; j < 4; j++)  
            cout << twoDimArray[i][j];  
  
    cout << endl;  
    return 0;  
}
```



C++ also supports the creation of multidimensional arrays, through the addition of more than one set of brackets. Thus, a two-dimensional array may be created by the following: *type arrayName[dimension1][dimension2];*

The array can also be initialized at declaration in the following ways:

```
int twoDimArray[2][4] = { 6, 0, 9, 6, 2, 0, 1, 1 };
```

```
int twoDimArray[2][4] = { { 6, 0, 9, 6 }, { 2, 0, 1, 1 } };
```

Note that dimensions must *always* be provided when initializing multidimensional arrays, as it is otherwise impossible for the compiler to determine what the intended element partitioning is. For the same reason, when multidimensional arrays are specified as arguments to functions, all dimensions but the first *must* be provided (the first dimension is optional), as in the following:

```
int aFunction(int arr[][4]) { ... }
```

Declaring `int arr[2][4];` is the same thing as declaring `int arr[8];`.

Strings

String literals such as “Hello, world!” are actually represented by C++ as a sequence of characters in memory. In other words, a string is simply a character array and can be manipulated as such.

Consider the following program:

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

int main() {
    char helloworld[] = { 'H', 'e', 'l', 'l', 'o', ',', ' ',
                          'w', 'o', 'r', 'l', 'd', '!', '\0' };

    cout << helloworld << endl;

    return 0;
}
```

This program displays *Hello, world!* **Note that the character array helloworld ends with a special character known as the *null character*.** This character is used to indicate the end of the string.

Strings

Character arrays can also be initialized using string literals. In this case, no null character is needed, as the compiler will automatically insert one:

The individual characters in a string can be manipulated either directly by the programmer or by using special functions provided by the C/C++ libraries. These can be included in a program through the use of the `#include` directive. Of particular note are the following:

- `cctype` (`ctype.h`): character handling
- `cstdio` (`stdio.h`): input/output operations
- `cstdlib` (`stdlib.h`): general utilities
- `cstring` (`string.h`): string manipulation

Strings

```
using namespace std;

#include <iostream>
#include <cctype>
using namespace std;

int main() {
    char messyString[] = "t6H0I9s6.iS.999a9.STRING";

    char current = messyString[0];
    for(int i = 0; current != '\0'; current = messyString[++i]) {
        if(isalpha(current))
            cout << (char)(isupper(current) ? tolower(current) : current);
        else if(ispunct(current))
            cout << ' ';
    }

    cout << endl;
    return 0;
}
```

this is a string

This example uses the **isalpha**, **isupper**, **ispunct**, and **tolower** functions from the **cctype library**. The **is-** functions check whether a given character is an alphabetic character, an uppercase letter, or a punctuation character, respectively. These functions return a Boolean value of either true or false. The **tolower** function converts a given character to lowercase.

The for loop takes each successive character from **messyString** until it reaches the null character. On each iteration, if the current character is alphabetic and uppercase, it is converted to lowercase and then displayed. If it is already lowercase it is simply displayed. If the character is a punctuation mark, a space is displayed. All other characters are ignored. The resulting output is *this is a string*.

Strings

```
#include <iostream>
#include <iostream>
#include <cstring>
using namespace std;

int main() {
    char fragment1[] = "Nadege is a s";
    char fragment2[] = "tring!";
    char fragment3[20];
    char finalString[20] = "";

    strcpy(fragment3, fragment1);
    strcat(finalString, fragment3);
    strcat(finalString, fragment2);

    cout << finalString;
    return 0;}
```

```
Nadege is a string!
```

This example creates and initializes two strings, `fragment1` and `fragment2`. `fragment3` is declared but not initialized. `finalString` is partially initialized (with just the null character). `fragment1` is copied into `fragment3` using **`strcpy`**, in effect initializing `fragment3`, **`strcat`** is then used to concatenate `fragment3` onto `finalString` (the function overwrites the existing null character), thereby giving `finalString` the same contents as `fragment3`. Then **`strcat`** is used again to concatenate `fragment2` onto `finalString`. `finalString` is displayed, giving `Nadege is a string!`.

You are encouraged to read the documentation on these and any other libraries of interest to learn what they can do and how to use a particular function properly.

Strings

```
// null-terminated sequences of characters
#include <iostream>
using namespace std;

int main ()
{
    char question[] = "Please, enter your first
name: ";
    char greeting[] = "Hello, ";
    char yourname [80];
    cout << question;
    cin >> yourname;
    cout << greeting << yourname << "!";
    return 0;
}
```

```
Please, enter your first name: John
Hello, John!
```

As you can see, we have declared three arrays of char elements. The first two were initialized with string literal constants, while the third one was left uninitialized. In any case, we have to specify the size of the array: in the first two (question and greeting) the size was implicitly defined by the length of the literal constant they were initialized to. While for your name we have explicitly specified that it has a size of 80 chars.

Strings

Finally, sequences of characters stored in char arrays can easily be converted into string objects just by using the assignment operator:

```
string mystring;  
char myntcs[]="some text";  
mystring = myntcs;
```

```
You are called Natacha GOMOKO.
```

```
#include <iostream>  
#include <string>  
using namespace std;  
int main()  
{  
    string surname("Natacha");  
    string name("GOMOKO");  
    string total; //an empty string  
    total += surname; //we add the surname to it  
    total += " "; //then we add space  
    total += name; //and finally the family name  
    cout << "You are called " << total << "." << endl;  
    return 0;  
}
```

Unlike arrays, we can add several strings by using +=

Arrays: Dynamic arrays (Vectors)

So far, we have been dealing with arrays that have a fixed size, dynamic arrays are arrays whose sizes can vary. To deal with such, we should first start by including the line: **#include <vector>**

Due to this line, we at times refer to dynamic arrays as vectors.

The second step is to declare the array as follows: **vector<Type>Name(Dimension);**

```
#include <iostream>
#include <vector> //Do not forget !
using namespace std;
int main()
{
    vector<int> tab(5);
    return 0;
}
```

Arrays: Dynamic arrays (Vectors)

```
#include <iostream>
#include <vector> //Do not forget !
using namespace std;
int main()
{
    vector<int> table(5, 3); //creates a array of 5 integers each with value 3

    vector<string> listName(12, "No name"); //creates a array of 12 strings each with "No name"

    vector<double> tab; //creates an empty array
    return 0;
}
```

Arrays: Dynamic arrays (Vectors)

```
#include <iostream>
#include <vector> //Do not forget !
using namespace std;
int main()
{
    int const numberOfBestScores=5; //size of the array
    vector<int> bestScores(numberOfBestScores); //Declaration of array
    bestScores[0] = 118218; //filling the first entry
    bestScores[1] = 100432; //filling the second entry
    bestScores[2] = 87347;
    bestScores[3] = 64523;
    bestScores[4] = 31415;
    return 0;
}
```

Arrays: Dynamic arrays (Vectors)

If we wish to add an element at the end of the array, we use the function `push_back()`.

```
#include <iostream>
#include <vector> //Do not forget !
using namespace std;
int main()
{
    vector<int> tab(3,2); // an array of 3 elements of value 2
    tab.push_back(8); //we add a 4th element of value 8

    tab.push_back(7); //we add a 5th element of value 7

    tab.push_back(14); //we add a 6th element of value 14

    //the array now has as: 2 2 2 8 7 14
    return 0;
}
```

Arrays: Dynamic arrays (Vectors)

If we wish to delete the last entry of an array, we use the function `pop_back()`, but there brackets are empty

```
#include <iostream>
#include <vector> //Do not forget !
using namespace std;
int main()
{
    vector<int> tab(3,2); // an array of 3 elements of value 2

    tab.pop_back(); // 2 elements left
    tab.pop_back(); // 1 element left
    return 0;
}
```

Arrays: Dynamic arrays (Vectors)

Since the size of the array can change, we do not know exactly the number of elements in the array.

To solve this problem, we use the function `size ()`.

By using `tab.size()`, we obtain the integer corresponding to the number of elements in an array

```
#include <iostream>
#include <vector> //Do not forget !
using namespace std;
int main()
{
    vector<int> tab(5,4); // an array of 5 elements of value 4

    int const dim = tab.size();

    return 0;
}
```

Arrays: Dynamic arrays (Vectors)

Example:

```
#include <iostream>
#include <vector> //Do not forget !
using namespace std;
int main()      //we wish to calculate the average mark !
{
    vector<double> marks; //an empty array
    marks.push_back(12.5); //we add a 1st entry
    marks.push_back(19.5); //we add a 2nd entry
    marks.push_back(6);
    marks.push_back(12);
    marks.push_back(14.5);
    marks.push_back(15);

    double average(0);

    for(int i(0); i<marks.size(); ++i){
        average += marks[i]; }

    average /= marks.size();
    cout << "Your average is : " << average << endl;
    return 0;}
```

Your average is : 13.25

Arrays: Dynamic arrays (Vectors)

Case of functions:

```
//a function receiving an array of integers as arguments
void funct(vector<int> a)
{
    //...

//a function receiving an array of integers as arguments
void funct(vector<int> const& a)
{
    //...
}
```


Summary

- An array is a succession of variables in memory. A 4 dimensional array corresponds to 4 successive variables in memory. An array is initialised by `int bestScore[4];` (for 4 entries).
- The first entry is numbered 0 (`bestScore[0]`).
- If the size of the array is susceptible to vary, create a dynamic array of type `vector` :
`vector<int>tab(5);`
- We can create multidimensional arrays e.g. `int tableau[5][4];` means we have created an array of 5 rows and 4 columns.
- String characters can be considered as arrays. Each entry corresponds to a character

Exercise: Write a program that can calculate the average of numbers in an array

Reading and modifying files