Fundamentals of Programming I

5 Structured Data Types

Grado en Ingeniería Informática

Luis Hernández Yáñez
Facultad de Informática
Universidad Complutense

Index

Luis Hernández Yáñez/Pablo Moreno Ger

Data Types

Luis Hernández Yáñez/Pablo Moreno Ger

Data Types

Classification of types

- ✓ Simple Data Types
  - Standard: `int`, `float`, `double`, `char`, `bool` ✓
    Predefined set of values
  - Declared by the user: *enumerated* types ✓
    Set of values defined by the Programmer
- ✓ Structured Data Types
  - Homogeneous collections: *arrays* ✓
    All the elements of the same type
  - Heterogeneous collections: *structures*
    Elements with different types

Luis Hernández Yáñez/Pablo Moreno Ger

Structured Data Types

Collections (agglomerated types)

Data grouping (several elements):

- ✓ All of the same type: *array* (*table*)
- ✓ Different types: *structure* (*register*, *tuple*)

Arrays (tables)

- ➢ Elements organized by position (index): 0, 1, 2, 3, ...
- ➢ Access by index: 0, 1, 2, 3, ...
- ➢ One or several dimensions

Structures (tuples, registers)

- ➢ Elements (*fields*) in any order
- ➢ Access by name

Luis Hernández Yáñez/Pablo Moreno Ger

Fundamentals of Programming I

Arrays Revisited

Luis Hernández Yáñez/Pablo Moreno Ger

Arrays

Sequential structure

Each element in a certain position (*index*):

- ✓ Indexes are positive integers
- ✓ The index of the first element is ALWAYS 0
- ✓ Indexes are incremented by one

sales	125.40	76.95	328.80	254.62	435.00	164.29	0.00
	0	1	2	3	4	5	6

Direct access

Each element is accessed by its index:

`sales[4]` accesses the 5th element (it contains the value `435.00`)

```
cout << sales[4];
sales[4] = 442.75;
```

[ ]

Data of the same base type:
Used like any other variable

Luis Hernández Yáñez/Pablo Moreno Ger

Array Types and Variables

Array type declaration

```
const int Dimension = ...;
typedef base_type tName[Dimension];
```

Example:

```
const int Days = 7;
typedef double tSales[Days];
```

Array types variable declaration: like any other variable

```
tSales sales;
```

Elements ARE NOT automatically initialized!

It's the programmer's responsibility to use valid indexes!

Arrays can't be copied directly ~~`array1 = array2`~~

They must be copied element by element

Luis Hernández Yáñez/Pablo Moreno Ger

Arrays and for Loops

Processing arrays...

- ✓ Traversals
- ✓ Searches
- ✓ Sorts

Etcetera...

Array traversal with for loop

Arrays: fixed size → Fixed number of repetition loop (for)

```
tSales sales;
double mean, total = 0;
...
for (int i = 0; i < Days; i++)
   total = total + sales[i];
mean = total / Days;
```

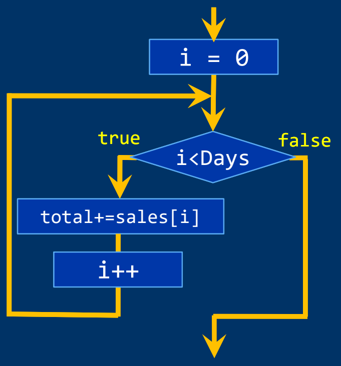
```
const int Days = 7;
typedef double tSales[Days];
```

Luis Hernández Yáñez/Pablo Moreno Ger

Arrays and for Loops

12.40	10.96	8.43	11.65	13.70	13.41	14.07
0	1	2	3	4	5	6

```
tSales sales;
double mean, total = 0;
...
for (int i = 0; i < Days; i++)
   total = total + sales[i];
```



...

Memory

Days	7
sales[0]	12.40
sales[1]	10.96
sales[2]	8.43
sales[3]	11.65
sales[4]	13.70
sales[5]	13.41
sales[6]	14.07
mean	?
total	84.62
i	7

Luis Hernández Yáñez/Pablo Moreno Ger

Fundamentals of Programming I

More About Arrays

Luis Hernández Yáñez/Pablo Moreno Ger

Array Initialization

Arrays can be initialized at declaration

Assign a value series to the array name:

```
const int DIM = 10;
typedef int tTable[DIM];
tTable t = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

Values are assigned in order:

```
t[0] t[1] t[2] t[3] t[4] ... t[9]
  ↑    ↑    ↑    ↑    ↑         ↑
  1st  2nd  3rd  4th  5th  ...  10th
```

If there are fewer values than elements, `0` is assigned to the rest

```
tTable t = { 0 }; // All the elements to 0
```

Luis Hernández Yáñez/Pablo Moreno Ger

Enumerated Types as Indexes

```
const int Colors = 3,
typedef enum { red, green, blue } tRGB;
typedef int tColor[Colors];
tColor color;
...
cout << "Amount of red (0-255): ";
cin >> color[red];
cout << "Amount of green (0-255): ";
cin >> color[green];
cout << "Amount of blue (0-255): ";
cin >> color[blue];
```

Remember that internally integers are used, starting with 0, for the different symbols of the enumerated type
red ≡ 0   green ≡ 1   blue ≡ 2

Luis Hernández Yáñez/Pablo Moreno Ger

Passing Arrays to Subprograms

Simulation of parameter passing by reference

Without using & in parameter declaration

Subprograms receive the array's memory address

```
const int Max = 10;
typedef int tTable[Max];
void initialize(tTable table); // Without &
```

Changes in array parameter are reflected in the argument

```
initialize(array);
```

If `initialize()` modifies any element of `table`, those elements of `array` are automatically modified

It is the same array!

Luis Hernández Yáñez/Pablo Moreno Ger

Passing Arrays to Subprograms

```
const int Dim = 10;
typedef int tTable[Dim];
void initialize(tTable table); // Without &

void initialize(tTable table) {
   for (int i = 0; i < Dim; i++)
      table[i] = i;
}
int main() {
   tTable array;
   initialize(array); // array is modified
   for (int i = 0; i < Dim; i++)
      cout << array[i] << " ";
   ...
```

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

Luis Hernández Yáñez/Pablo Moreno Ger

Passing Arrays to Subprograms

How to avoid changes in the array?

Using `const` modifier in parameter declaration:

`const tTable table` An array of constants

`void print(const tTable table);`

The argument will be treated as an array of constants

If in the subprogram there is any instruction that tries to modify any element in the array: *compilation error!*

```
void print(const tTable table) {
   for (int i = 0; i < Dim; i++)
      cout << table[i] << " ";
      // OK. It is accessed, but not modified
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

List Implementation

Luis Hernández Yáñez/Pablo Moreno Ger

List Implementation with Arrays

Lists with a fixed number of elements

Array with the number of elements as the dimension

```
const int NUM = 100;
typedef double tList[NUM];
tList list; // Exactly 100 doubles
```

List traversal:

```
for (int i = 0; i < NUM; i++) {
   ...
```

Searching in the list:

```
while ((i < NUM) && !found) {
   ...
```

Luis Hernández Yáñez/Pablo Moreno Ger

List Implementation with Arrays

Lists with a variable number of elements

Array with the maximum number of elements + Element counter

```
const int MAX = 100;
typedef double tList[MAX];
tList list; // Up to 100 elements
int counter = 0;
```

Array and counter unbound? → Structures

List traversal:

```
for (int i = 0; i < counter; i++) {
   ...
```

Searching in the list:

```
while ((i < counter) && !found) {
   ...
```

Luis Hernández Yáñez/Pablo Moreno Ger

Fundamentals of Programming I

Character Strings

Luis Hernández Yáñez/Pablo Moreno Ger

Character Strings

Character arrays

String: character sequence of variable length

```
"Hello" "Bye" "Supercalifragilistic" "1234 56 7"
```

String variables: contain character sequences

Stored in character arrays: maximum length (dimension)

Not all the array elements are relevant:

✓ String length: number of characters, from the beginning, that actually constitute the string:

H	e	l	l	o																	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Current length: 5

Luis Hernández Yáñez/Pablo Moreno Ger

Character Strings

String length

B	y	e																			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Length: 3

S	u	p	e	r	c	a	l	i	f	r	a	g	i	l	i	s	t	i	c		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Length: 20

We need to know where the relevant characters end:

✓ Keep string length as associated data

✓ Use a termination character at the end (*sentinel*)

B	y	e	\0							
0	1	2	3	4	5	6	7	8	9	10

Luis Hernández Yáñez/Pablo Moreno Ger

Character Strings

Character strings in C++

Two alternatives for string implementation:

- ✓ C style strings (*null terminated*)
- ✓ Type `string`

C style strings *Lesson Supplement*

- ✓ Arrays of type `char` with a maximum length
- ✓ Special character at the end: `'\0'`

Type `string`

- ✓ More sophisticated strings
- ✓ No maximum length (automatic memory management)
- ✓ Many utility subprograms (`string` Library)

Luis Hernández Yáñez/Pablo Moreno Ger

Fundamentals of Programming I

`string`-type Character Strings

Luis Hernández Yáñez/Pablo Moreno Ger

string-type Character Strings

string *Type*

- ✓ Assumes memory management responsibility
- ✓ Defines overloaded operators (i.e., + for appending)
- ✓ Safer and more efficient strings

`string` library

Requires using namespace `std`

- ✓ They are automatically initialized to an empty string
- ✓ They can be initialized at declaration
- ✓ They can be copied with the assignment operator
- ✓ They can be concatenated with + operator
- ✓ There are many utility routines

Luis Hernández Yáñez/Pablo Moreno Ger

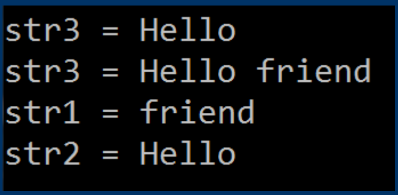
string-type Strings

string.cpp

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

int main() {
   string str1("Hello");   // initialization
   string str2 = "friend"; // initialization
   string str3;
   str3 = str1; // copy
   cout << "str3 = " << str3 << endl;
   str3 = str1 + " "; // concatenation
   str3 += str2;      // concatenation
   cout << "str3 = " << str3 << endl;
   str1.swap(str2);  // interchange
   cout << "str1 = " << str1 << endl;
   cout << "str2 = " << str2 << endl;

   return 0;
}
```



```
str3 = Hello
str3 = Hello friend
str1 = friend
str2 = Hello
```

Luis Hernández Yáñez/Pablo Moreno Ger

`string`-type Strings

Length of the string:

`str.length()` or `str.size()`

We can compare two strings with relational operators:

```
if (str1 <= str2) { ...
```

String character access:

- ✓ As a simple character array: *`str`*`[`*`i`*`]`
  No access control for invalid array indexes
  To be used only with indexes absolutely known to be valid
- ✓ Function `at(`*`index`*`)`: *`str`*`.at(`*`i`*`)`
  Execution error if an inexistent position is tried to be accessed

Luis Hernández Yáñez/Pablo Moreno Ger

I/O with `string`-type Strings

- ✓ Displayed on the screen with `cout <<`
- ✓ Input with `cin >>` Skips leading blank space
  Input ends with next blank space (incl. Enter)
- ✓ To discard the rest of characters in the buffer:
  `cin.sync();` or `cin.ignore()`
- ✓ Input including blank spaces:
  `getline(cin,` *`string`*`)`
  Assigns to the *`string`* characters read until the end of the line
  If input is pending at Enter, then an empty string will be read
- ✓ Reading from text files:
  Same as with console; `sync()` has no effect
  `file >> str` `getline(file, str)`

Luis Hernández Yáñez/Pablo Moreno Ger

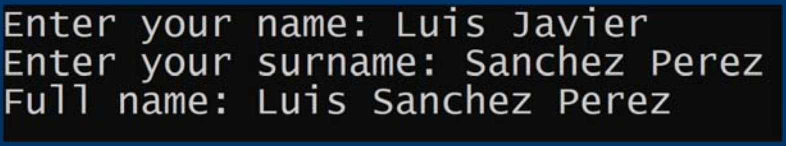
I/O with `string`-type Strings

string2.cpp

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

int main() {
   string name, surname;
   cout << "Enter your name: ";
   cin >> name; // Input ends with the first space or Enter
   cout << "Enter your surname: ";
   cin.sync();
   getline(cin, surname); // Input ends with Enter
   cout << "Full name: " << name << " " << surname << endl;

   return 0;
}
```



```
Enter your name: Luis Javier
Enter your surname: Sanchez Perez
Full name: Luis Sanchez Perez
```

Luis Hernández Yáñez/Pablo Moreno Ger

Operations with `string`-type Strings

✓ *`str.substr(position, length)`*
Substring of *length* characters from *position* (starting at 0)

```
string str = "abcdefg";
cout << str.substr(2, 3); // Prints cde
```

✓ *`str.find(substring)`*
Position of first occurrence of *substring* in *str*

```
string str = "Olala";
cout << str.find("la"); // Prints 1
```

(Remember that character array indexes start at 0)

✓ *`str.rfind(substring)`*
Position of last occurrence of *substring* in *str*

```
string str = "Olala";
cout << str.rfind("la"); // Prints 3
```

Luis Hernández Yáñez/Pablo Moreno Ger

Operations with `string`-type Strings

- ✓ *`str.erase(from, num)`*
  Erases *num* characters from position *from*

```
string str = "abcdefgh";
str.erase(3, 4); // str now contains "abch"
```

- ✓ *`str.insert(where, str2)`*
  Inserts *str2* in position *where*

```
string str = "abcdefgh";
str.insert(3, "123"); // str now contains "abc123defgh"
```

http://www.cplusplus.com/reference/string/string/

Luis Hernández Yáñez/Pablo Moreno Ger

Fundamentals of Programming I

Structures

Luis Hernández Yáñez/Pablo Moreno Ger

Structures

Heterogeneous collections (tuples, registers)

Elements of (possibly) different types: *fields*

Fields identified by name

Related information that can be managed as a unit

Each element is accessed with its name (`.` operator)

Luis Hernández Yáñez/Pablo Moreno Ger

Structure Types

```
struct tType { // type name: after struct!
   ... // field declaration (like variables)
};
```

In the last revision of the C++ language, `typedef` is not used with `struct`

```
struct tPerson {
   string name;
   string surname;
   int age;
   string nif;
};
```

Fields: Any known type (standard or previously declared)

Luis Hernández Yáñez/Pablo Moreno Ger

Structure Variables

```
tPerson person;
```

Variables of type `tPerson` contain four data (fields):

```
name     surname      age      nif
```

Access to fields with dot operator (`.`):

```
  person.name // a string
  person.surname // a string
  person.age    // an integer
  person.nif // a string
```

We can copy two structures directly:

```
tPerson person1, person2;
...
person2 = person1;
```

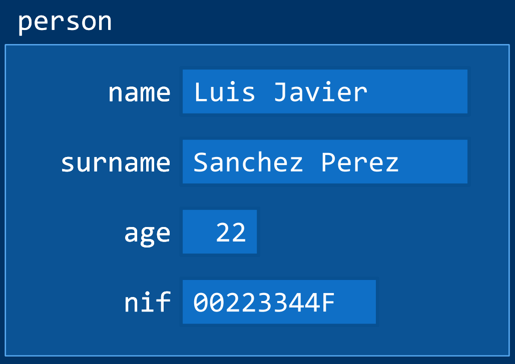
All fields are copied (*even if they are arrays!*)

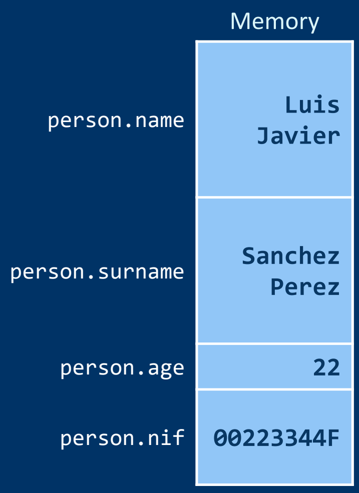
Luis Hernández Yáñez/Pablo Moreno Ger

BY NC SA

Heterogeneous Data Groupings

```
struct tPerson {
   string name;
   string surname;
   int age;
   string nif;
};
tPerson person;
```





Luis Hernández Yáñez/Pablo Moreno Ger

BY NC SA

Elements in any Order

```
struct tPerson {
   string name;
   string surname;
   int age;
   string nif;
};
tPerson person;
```

Fields are not arranged in a specific order

Direct access with field name (operator `.`)

Each field can be used like any other variable of its type

Initial values for the fields:

```
struct tPerson {
   string name;
   string surname;
   int age = 0;
   string nif;
};
```

Each time a `tPerson` is created the field `age` gets the value `0`

Structures are passed by value (without &) or by reference (with &) to subprograms

Luis Hernández Yáñez/Pablo Moreno Ger

Structures Within Structures

```
struct tNif {                    struct tPerson {
   string dni;                      ...
   char letter;                     tNif nif;
};                               };
```

```
tPerson person;
```

Access to entire NIF:

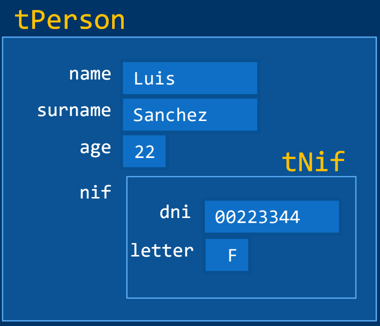
```
person.nif // Another structure
```

Access to NIF's letter:

```
person.nif.letter
```

Access to DNI:

```
person.nif.dni
```



Luis Hernández Yáñez/Pablo Moreno Ger

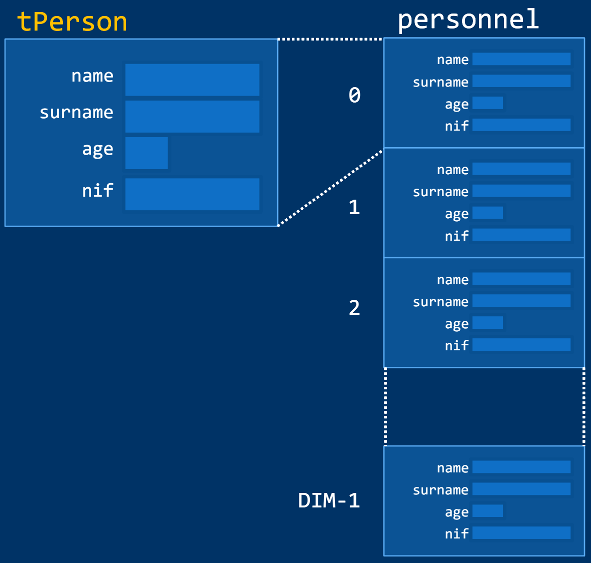
Arrays of Structures

```
struct tPerson {
   string name;
   string surname;
   int age;
   string nif;
};
const int DIM = 100;
typedef tPerson tArray[DIM];
tArray personnel;
```

Name of the third person:
`personnel[2].name`

Age of the twelfth person:
`personnel[11].age`

NIF of the first person:
`personnel[0].nif`



Luis Hernández Yáñez/Pablo Moreno Ger

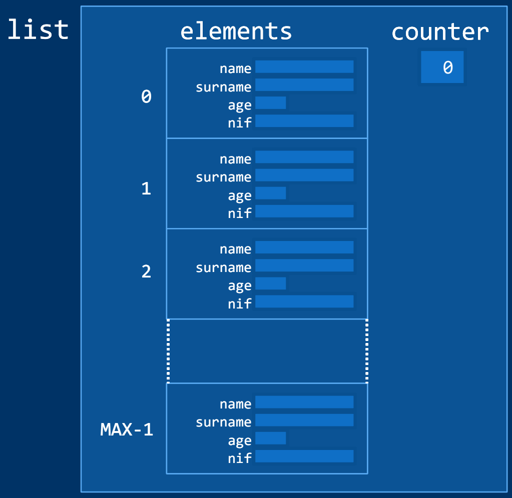
Arrays Within Structures

```
const int MAX = 100;
struct tPerson {
   string name;
   string surname;
   int age;
   string nif;
};
typedef tPerson tArray[MAX];
struct tList {
   tArray elements;
   int counter = 0;
};
tList list;
```

Name of the third person: `list.elements[2].name`

Age of the twelfth person: `list.elements[11].age`

NIF of the first person: `list.elements[0].nif`



Luis Hernández Yáñez/Pablo Moreno Ger

Fundamentals of Programming I

Variable Length Lists

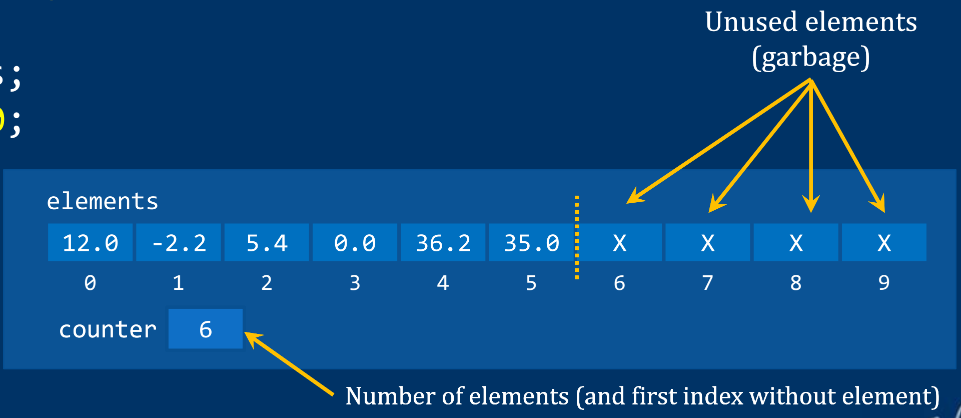
Luis Hernández Yáñez/Pablo Moreno Ger

cc BY NC SA

Variable Length Lists

Structure grouping the array and the counter:

```
const int MAX = 10;
typedef double tArray[MAX];
struct tList {
   tArray elements;
   int counter = 0;
};
```


Unused elements (garbage)
elements
12.0
-2.2
5.4
0.0
36.2
35.0
X
X
X
X
0
1
2
3
4
5
6
7
8
9
counter
6
Number of elements (and first index without element)

Main operations: element insertion and deletion

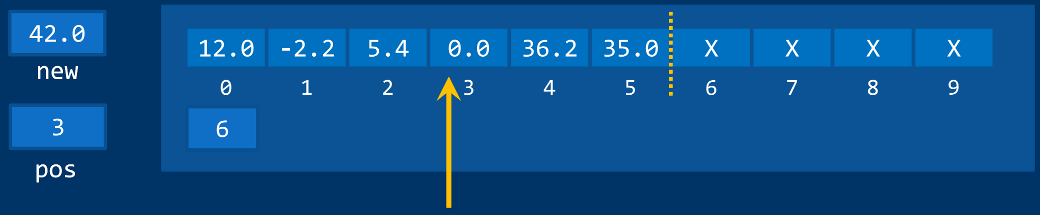
Luis Hernández Yáñez/Pablo Moreno Ger

cc BY NC SA

Element Insertion

Inserting a new element in a certain position

Valid positions: 0 to `counter`



Make sure there is place for more elements (counter < maximum)

3 step operation:

1.- Make room for the new element (from the position)

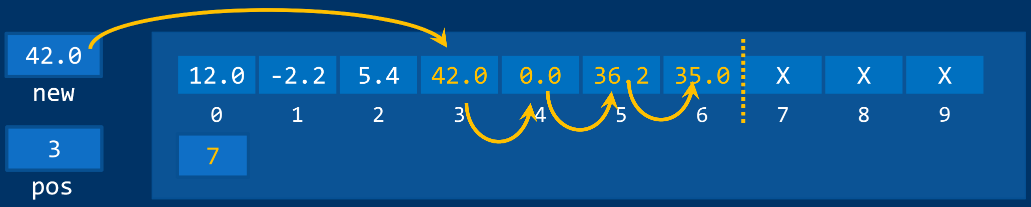
2.- Place new element in the position

3.- Increase counter by one

Luis Hernández Yáñez/Pablo Moreno Ger

Element Insertion

```
if (list.counter < MAX) {
   // Make room...
   for (int i = list.counter; i > pos; i--)
      list.elements[i] = list.elements[i - 1];
   // Insert and increment counter
   list.elements[pos] = newElement;
   list.counter++;
}
```

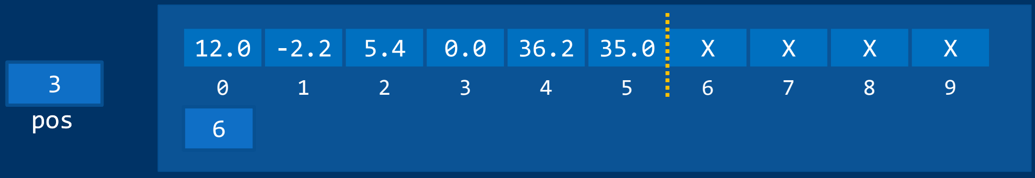


Luis Hernández Yáñez/Pablo Moreno Ger

Element Deletion

Deleting the element in a certain position

Valid positions: 0 to counter-1



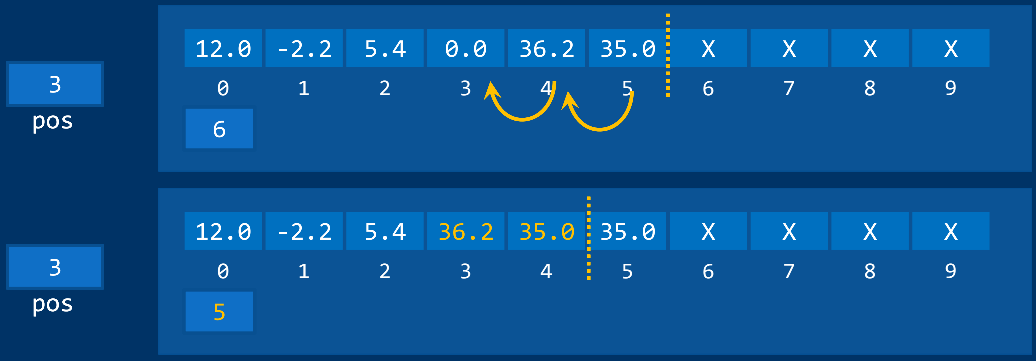
Shift one position to the left of next one and decrease counter by one:

```
for (int i = pos; i < list.counter - 1 ; i++)
   list.elements[i] = list.elements[i + 1];
list.counter--;
```

Luis Hernández Yáñez/Pablo Moreno Ger

Element Deletion

```
for (int i = pos; i < list.counter - 1 ; i++)
   list.elements[i] = list.elements[i + 1];
list.counter--;
```



Luis Hernández Yáñez/Pablo Moreno Ger

A Complete Example

Luis Hernández Yáñez/Pablo Moreno Ger

Variable Length List Example

Description

Program to manage a list with the students in a class

For each student: name, surname, age, NIF and grade

- ✓ The total number of students is not known (up to 100)
- ✓ List information is maintained in a file `class.txt`

  It will be loaded at the beginning and saved at the end
- ✓ The program should offer these actions to the user:
  - — Add a new student
  - — Delete an existing student
  - — Grade all the students
  - — Get a listing with grades, identifying the highest and displaying the average

Luis Hernández Yáñez/Pablo Moreno Ger

Variable Length List Example

db.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>
using namespace std;

const int MAX = 100;
struct tStudent {
   string name;
   string surname;
   int age;
   string nif;
   double grade;
};
```

```
typedef tStudent tArray[MAX];
struct tList {
   tArray elements;
   int counter = 0;
};
```

The list counter is automatically initialized to 0!

Global constants and types after the libraries

Luis Hernández Yáñez/Pablo Moreno Ger

Variable Length List Example

```
// Prototypes
int menu(); // Program menu – Returns chosen (valid) option
void load(tList &list, bool &ok); // Loads the info in the file into the list
void save(const tList &list); // Saves the list in the file
void readStudent(tStudent &student); // Reads info for one student
void insertStudent(tList &list, tStudent student, bool &ok);
// Inserts a new student in the list
void deleteStudent(tList &list, int pos, bool &ok);
// Deletes the student in that position
string fullName(tStudent student); // name, space and surname
void goGrade(tList &list); // Grade the students
double classMean(const tList &list); // Average grade
int highestGrade(const tList &list); // Index of student with highest grade
void printStudent(tStudent student);
void listing(const tList &list, double mean, int highest); // Class listing
```

Prototypes after global declarations

Luis Hernández Yáñez/Pablo Moreno Ger

Fundamentals of Programming I

do-while Loop

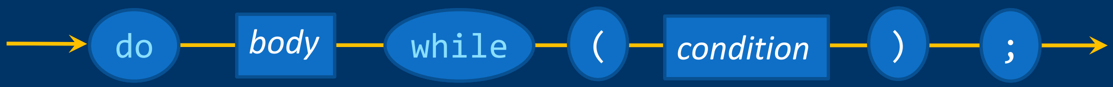
Luis Hernández Yáñez/Pablo Moreno Ger

Another C++ Indeterminate Loop

do..while *Loop*

`do` *body* `while (`*condition*`);`

Condition at loop's end



```
int i = 1;
do {
   cout << i << endl;
   i++;
} while (i <= 100);
```

body is always executed at least once

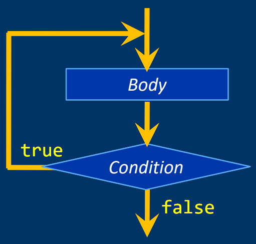
body is a code block

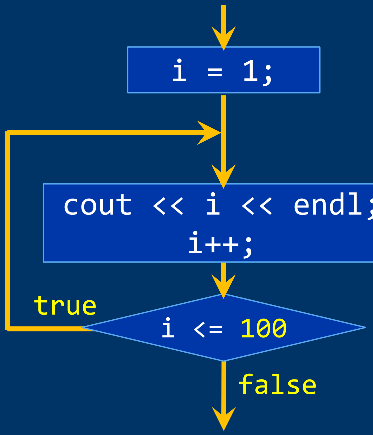
Luis Hernández Yáñez/Pablo Moreno Ger

do-while Loop Execution

```
int i = 1;
do {
   cout << i << endl;
   i++;
} while (i <= 100);
```

Body
is executed
at least once





Luis Hernández Yáñez/Pablo Moreno Ger

while *versus* do-while

Does the loop's body have to be executed at least one time?

```
cin >> d; // First reading
while (d != 0) {
   sum = sum + d;
   count++;
   cin >> d;
}
```

```
do {
   cin >> d;
   if (d != 0) { // Final?
      sum = sum + d;
      count++;
   }
} while (d != 0);
```

```
cout << "Option: ";
cin >> op; // First reading
while ((op < 0) || (op > 4)) {
   cout << "Option: ";
   cin >> op;
}
```

```
do { // Simpler
   cout << "Option: ";
   cin >> op;
} while ((op < 0) || (op > 4));
```

Luis Hernández Yáñez/Pablo Moreno Ger

Program Menu with do-while

```
int menu() {
   int op;

   do {
      cout << "1 - Add a new student" << endl;
      cout << "2 - Delete a student" << endl;
      cout << "3 - Grade students" << endl;
      cout << "4 - Class listing" << endl;
      cout << "0 - Exit" << endl;
      cout << "Option: ";
      cin >> op;
   } while ((op < 0) || (op > 4));

   return op;
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Variable Length List Example

The file `class.txt`

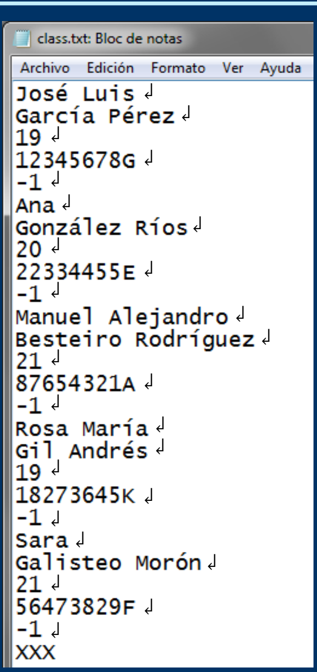
One piece of data in each line

For each student:

- ✓ Name (string)
- ✓ Surname (string)
- ✓ Age (integer)
- ✓ NIF (string)
- ✓ Grade (real number; -1 if not graded)

Ends with XXX as name

The file is supposed to be correct



```
José Luis
García Pérez
19
12345678G
-1
Ana
González Ríos
20
22334455E
-1
Manuel Alejandro
Besteiro Rodríguez
21
87654321A
-1
Rosa María
Gil Andrés
19
18273645K
-1
Sara
Galisteo Morón
21
56473829F
-1
XXX
```

Luis Hernández Yáñez/Pablo Moreno Ger

Variable Length List Example

Reading student information

Name and surname: There may be several words → `getline()`

Age → Extractor (>>)

NIF: One word → Extractor (>>)

Grade → Extractor (>>)

The Enter after the grade number keeps pending
We should skip (read and discard) that character with `get()`
If not so, the next name read will get an empty string (Enter)

Don't read directly in the list:
~~`getline(file, list.elements[list.counter].name);`~~
Read in a temporary variable of type `tStudent`

Luis Hernández Yáñez/Pablo Moreno Ger

Loading the List from the File `class.txt`

```
void load(tList &list, bool &ok) {
   tStudent student; // Temporary variable to read in
   ifstream file;
   char aux;
   file.open("class.txt");
   if (!file.is_open())
      ok = false;
   else {
      ok = true;
      getline(file, student.name); // Reads the first name
      while ((student.name != "XXX") && (list.counter < MAX)) {
         getline(file, student.surname);
         file >> student.age >> student.nif >> student.grade;
         file.get(aux); // Skips Enter
         list.elements[list.counter] = student; // At the end
         list.counter++;
         getline(file, student.name); // Next name
      } // If more than MAX students, we ignore the rest
      file.close();
   }
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Saving the List in the File class.txt

One piece of data in each line (name, surname, age, NIF, grade):

```
void save(const tList &list) {
   ofstream file;
   file.open("class.txt");
   for (int i = 0; i < list.counter; i++) {
      file << list.elements[i].name << endl;
      file << list.elements[i].surname << endl;
      file << list.elements[i].age << endl;
      file << list.elements[i].nif << endl;
      file << list.elements[i].grade << endl;
   }
   file << "XXX" << endl; // Final sentinel
   file.close();
}
```

`const tList &list` → Constant reference
Passed by reference but as a constant ≡ Passed by value
Avoids copying the argument into the parameter (big structures)

Luis Hernández Yáñez/Pablo Moreno Ger

Reading Student Data from the Keyboard

```
void readStudent(tStudent &student) {
   cin.sync(); // Discards any pending input
   cout << "Name: ";
   getline(cin, student.name);
   cout << "Surname: ";
   getline(cin, student.surname);
   cout << "Age: ";
   cin >> student.age;
   cout << "NIF: ";
   cin >> student.nif;
   student.grade = -1; // No grade yet
   cin.sync(); // Discards any pending input
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Inserting a New Student

```
void insertStudent(tList &list, tStudent student, bool &ok) {
   ok = true;
   if (list.counter == MAX)
      ok = false;
   else {
      list.elements[list.counter] = student;
      // Inserts at the end
      list.counter++;
   }
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Deleting a Student

```
void deleteStudent(tList &list, int pos, bool &ok) {
// Expects the element index in pos
   if ((pos < 0) || (pos > list.counter - 1))
      ok = false; // Inexistent element
   else {
      ok = true;
      for (int i = pos; i < list.counter - 1; i++)
         list.elements[i] = list.elements[i + 1];
      list.counter--;
   }
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Grading the Students

```
string fullName(tStudent student) {
   return student.name + " " + student.surname;
}

void goGrade(tList &list) {
   for (int i = 0; i < list.counter; i++) {
      cout << "Grade for the student "
         << fullName(list.elements[i]) << ": ";
      cin >> list.elements[i].grade;
   }
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

More subprograms

```
double classMean(const tList &list) {
   double total = 0.0;
   for (int i = 0; i < list.counter; i++)
      total = total + list.elements[i].grade;
   return total / list.counter;
}

int highestGrade(const tList &list) {
   double max = 0;
   int pos = 0;
   for (int i = 0; i < list.counter; i++)
      if (list.elements[i].grade > max) {
         max = list.elements[i].grade;
         pos = i;
      }
   return pos;
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

The Listing

```
void printStudent(tStudent student) {
   cout << setw(38) << left << fullName(student) << student.nif << " "
        << setw(2) << right << student.age << " " << fixed
        << setprecision(1) << student.grade;
}

void listing(const tList &list, double mean, int highest) {
   for (int i = 0; i < list.counter; i++) {
      cout << setw(3) << right << i + 1 << ": ";
      printStudent(list.elements[i]);
      if (i == highest)
         cout << " <<< Highest grade!";
      cout << endl;
   }
   cout << "Class average grade: " << fixed
        << setprecision(1) << mean << endl << endl;
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

The Main Program

```
int main() {
   tList list;
   tStudent student;
   bool success;
   int op, pos;

   load(list, success);
   if (!success)
      cout << "Couldn't open the file!" << endl;
   else {
      do { // do loop doesn't need to have read the first option
         op = menu();
         switch (op) {
         case 1:
            readStudent(student);
            insertStudent(list, student, success);
            if (!success)
                  cout << "List full: Impossible to insert!" << endl;
            break;
```

.../...

Luis Hernández Yáñez/Pablo Moreno Ger

The Main Program

```
         case 2:
            cout << "Position: ";
            cin >> pos;
            deleteStudent(list, pos - 1, success);
            if (!success)
               cout << "Inexistent element!" << endl;
            break;
         case 3:
            goGrade(list);
            break;
         case 4:
            listing(list, classMean(list), highestGrade(list));
         }
      } while (op != 0);
      save(list);
   }
   return 0;
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Promote Open Culture!

Creative Commons License

Attribution
You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Non commercial
You may not use this work for commercial purposes.

Share alike
If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

Click on the upper right image to learn more...

Luis Hernández Yáñez/Pablo Moreno Ger