

# Object-Oriented Programming

Iuliana Bocicor  
*iuliana@cs.ubbcluj.ro*

Babes-Bolyai University

2018

# Overview

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- 1 Pointers
- 2 Memory management
- 3 Modular programming in C/C++
- 4 Abstract Data Types - ADT
- 5 Summary

# Questions we will answer today

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- What is the difference between the stack and the heap?
- How can we allocate and free memory on the heap?
- How do we use pointers to access memory locations?
- How can we create modules in C? How do we separate the interface from the implementation?

## Recap

- Pointers are variables storing memory addresses.
- They allow us to manipulate data more flexibly.
- *Dereferencing* means accessing the value pointed to by a pointer.
- Dereferencing operator:  $*$ .
- Address operator:  $\&$ .

# Null and dangling pointers I

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

## Null pointer

- It is a pointer set to 0; there is no memory location 0  $\Rightarrow$  invalid pointer.
- Pointers are often set to 0 (or NULL) to signal that they are not currently valid.
- We should check whether a pointer is null before dereferencing it!

# Null and dangling pointers II

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

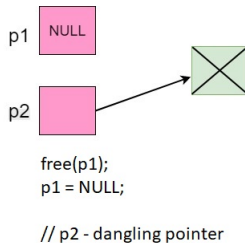
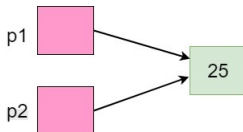
Abstract Data  
Types - ADT

Summary

## Dangling pointer

- It is a pointer that does not point to valid data:
  - the data might have been erased from memory;
  - the memory pointed to has undefined contents.
- Dereferencing such a pointer will lead to undefined behaviour!

# Null and dangling pointers III



## DEMO

Null and dangling pointers. (*NullDanglingPointers.c*).

# Arrays and pointers

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- Arrays can be seen as pointers to the first element of the array.
- `int arr[10];` **?** What is the difference between `arr` and `&arr[0]`?
- When passed as function parameters, arrays are passed "by reference".
- Please see the **Arrays.c** file in **Lecture1\_demo**.



# Pointers to functions I

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- a **function pointer** is a pointer which points to an address of a function;
- can be used for *dynamic (late)* binding (the function to use is decided at runtime, instead of compile time);
- functions can be used as parameters for other functions;
- do not need memory allocation/deallocation.

## Definition

`<return_type> (* <name>)(<parameter_types>)`

E.g.

```
double (*operation)(double, double);
```

# Pointers to functions II

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

## DEMO

Pointers to functions. (*PointersToFunctions.c*).

# Pointers to functions III

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

**C isn't that hard:**

**void (\*(f[]))()**

- declare f as array of pointers to functions returning pointers to functions returning void (<https://www.cdecl.org/>).

f	f
f[]	is an array
*f[]	of pointers
(*f[])	to functions
*(f[])	returning pointers
(*(f[]))()	to functions
void (*(f[]))();	returning void

# Const pointers

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- *Changeable pointer to constant data* - the pointed value cannot be changed, but the pointer can be changed to point to a different constant value.

```
const int* p;
```

- *Constant pointer to changeable data* - the pointed value can be changed through this pointer, but the pointer cannot be changed to point to a different memory location.

```
int* const p;
```

- *Constant pointer to constant data.*

```
const int* const p;
```

## DEMO

Const pointers. (*ConstPointers.c*).

# Stack and heap I

The memory used by a program is composed of several segments:

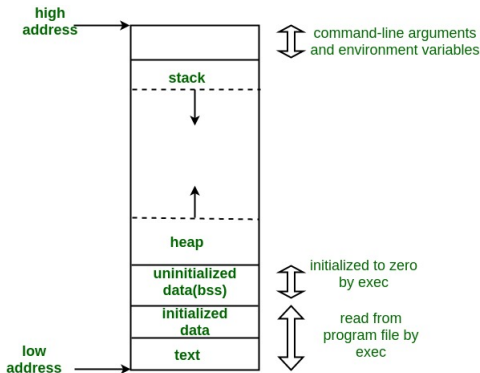


Figure source: <https://www.geeksforgeeks.org/memory-layout-of-c-program/>

# Stack and heap II

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- *The code (text) segment* - contains the compiled program.
- *The data segment* - used to store global and static variables (uninitialised variables are stored in the BSS segment).
- *The stack* - used to store function parameters, local variables and other function-related information.
- *The heap* - used for the dynamically allocated variables.

# Stack and heap III

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

## Stack



Figure source: <http://www.dreamstime.com/stock-photo-stack-books-white-background-image51790778>

## Heap

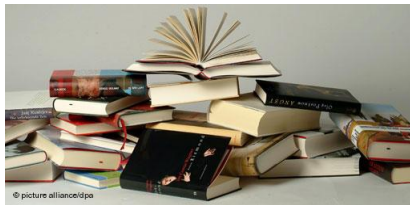


Figure source: <http://www.dw.com/en/digital-wave-threatens-germanys-fixed-price-book-world/a-5518440>

## Stack

- Is a continuous block of memory consisting of **stack frames**.
- **Stack frame** - keeps the data associated with one function call: return address, function arguments, local variables.
- For each function call, a new stack frame is constructed and pushed onto the stack.
- When a function is terminated, its associated stack frame is popped off the stack, the local variables are destroyed and execution is resumed at the return address.
- The stack has a limited size.
- ? Stack overflow



## Heap

- Large pool of memory.
- Used for dynamic memory allocation.
- The data in the heap must be managed by the programmer.
- The size of the heap is only limited by the size of the virtual memory.

# Memory management

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- Memory can be allocated in two ways:
  - *Statically* (compile time)
    - by declaring variables;
    - the size must be known at compile time;
    - there is no control over the lifetime of variables.
  - *Dynamically* ("on the fly", during run time)
    - on the heap;
    - the size does not have to be known in advance by the compiler;
    - is achieved using pointers;
    - the programmer controls the size and lifetime of the variables.

# Dynamic allocation and deallocation I

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- **C** - use the functions defined in **stdlib.h**:
  - **malloc** - finds a specified size of free memory and returns a void pointer to it (memory is uninitialised).
  - **calloc** - allocates space for an array of elements, initializes them to zero and then returns a void pointer to the memory.
  - **realloc** - reallocates the given area of memory (either by expanding or contracting or by allocating a new memory block).
  - **free** - releases previously allocated memory.

## DEMO

Dynamic allocation and deallocation in C. (*DynamicMemory-ManagementC.c*).

# Dynamic allocation and deallocation II

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- **C++** - **new** and **delete** operators.
- **new T**
  - memory is allocated to store a value of type T;
  - it returns the address of the memory location;
  - the return value has type T\*.
- **delete p**
  - deallocates memory that was previously allocated using **new**;
  - precondition: p is of type T\*;
  - the memory space allocated to the variable p is free.

## DEMO

Dynamic allocation and deallocation in C++. (*DynamicMemoryManagement.cpp*).

# Memory errors I

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- Memory leaks - memory is allocated, but not released (Visual Studio: `<crtdbg.h>` and `_CrtDumpMemoryLeaks();`).

```
Dumping objects ->
{99} normal block at 0x0000019A78F6BA50, 4 bytes long.
Data: < > 10 00 00 00
{98} normal block at 0x0000019A78F6BA10, 4 bytes long.
Data: < > 0E 00 00 00
{97} normal block at 0x0000019A78F6B9D0, 4 bytes long.
Data: < > 0C 00 00 00
{96} normal block at 0x0000019A78F6B990, 4 bytes long.
Data: < > 0A 00 00 00
{95} normal block at 0x0000019A78F64180, 4 bytes long.
Data: < > 08 00 00 00
{94} normal block at 0x0000019A78F64140, 4 bytes long.
Data: < > 06 00 00 00
{93} normal block at 0x0000019A78F64100, 4 bytes long.
Data: < > 04 00 00 00
{92} normal block at 0x0000019A78F656C0, 4 bytes long.
Data: < > 02 00 00 00
{91} normal block at 0x0000019A78F60830, 4 bytes long.
Data: < > 00 00 00 00
Object dump complete.
```

Memory  
leaks!

# Memory errors II

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- Invalid memory access - unallocated or deallocated memory is accessed.
- Mismatched Allocation/Deallocation - deallocation is attempted with a function that is not the logical counterpart of the allocation function used.
- Freeing memory that was never allocated.
- Repeated frees - freeing memory which has already been freed.

# So...when should we use pointers?

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- When data needs to be allocated on the heap ( ? when is that?).
- When we need "pass by reference".
- When we want to avoid copying data (because of the default "pass by value").
- For efficiency - to avoid copying data structures.
- ? Where are pointers allocated? Where are the objects pointed to by pointers allocated?

# Modules

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

A **module** is collection of functions and variables that implements a well defined functionality.

## Goals:

- separate the *interface* from the *implementation*;
- hide the implementation details.



# Header files. Libraries I

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- Function prototypes (function declarations) are grouped into a separate file called *header file*.
- A library is a set of functions, exposed for use by other programs.
- Libraries are generally distributed as:
  - a header file (.h) containing the function prototypes and
  - a binary file (.dll or .lib) containing the compiled implementation.
- The source code (.c/.cpp) does not need to be shared.

# Header files. Libraries II

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- The library users only need the function prototypes (which are in the header), not the implementation.
- The function specification is separated from the implementation.
- Static linking happens at compile time and the .lib is completely "included" in the executable (  $\Rightarrow$  an increase in the size of the resulting executable).
- Dynamic linking (.dll files) includes only the information needed at run time to locate and load the DLL that contains a data item or function.

# Preprocessor directives I

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- lines in the code preceded by a hash sign (#).
- are executed by the preprocessor, before compilation.

## Examples:

- `#include header_file` - tells the preprocessor to open the header file and insert its contents.
  - if the header file is enclosed between angle brackets (`<>`) - the file is searched in the system directories.
  - if the header is enclosed between double quotes ("`"`") - the file is first searched in the current directory and then in the system directories.

# Preprocessor directives II

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- `#define identifier replacement` - any occurrence of *identifier* in the code is replaced by *replacement*.
- `#ifdef macro, ... ,#endif` - the section of code between these two directives is compiled only if the specified macro has been defined.
- `#ifndef macro, ... ,#endif` - the section of code between these two directives is compiled only if the specified macro has **not** been defined.

# Preprocessor directives III

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- `#ifndef` `#define` and `#endif` - can be used as *include guards*.
- *include guards* are used to avoid *multiple inclusion* when using the `#include` directive. *Multiple inclusion* causes compilation errors (violation of the *One Definition Rule*).
- `#pragma` - used to specify various options to the compiler. `#pragma once` (not standard, but widely supported) - the current file will be included only once in a single compilation (same purpose as include guards).

# Create modular programs I

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

The code of a C/C++ program is split into several source files: .h and .c/.cpp:

- .h files - contain the function declarations (the interfaces);
- .c/.cpp files - contain the function implementations.

*Advantage:* the .c/.cpp files can be compiled separately (for error checking and testing).

- Whenever a header file is changed all the files that include it (directly or indirectly) must be recompiled.

# Create modular programs II

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- The header file is a **contract** between the developer and the client of the library that describes the data structures and states the arguments and return values for function calls.
- The compiler enforces the contract by requiring the declarations for all structures and functions before they are used (this is why the header file must be included).

# Module design guidelines I

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- Separate the interface from the implementation:
  - The header file should only contain type declarations and function prototypes.
  - Hide and protect the implementation details.
- Include a short description of the module (comment).
- Cohesion
  - A module should have a single responsibility.
  - The functions inside the module should be related.
- Layered architecture
  - Layers: model, validation, repository, controller, ui.
  - Manage dependencies: each layer depends only on the "previous" layer.



# Module design guidelines II

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

- Abstract data types (ADT)
  - Declare operations in the .h file and implement them in the .c/.cpp file.
  - Hide the implementation details, the client should only have access to the interface.
  - Abstract specification (functions' specifications should be independent from the implementation).
- Create self contained headers: they include all the modules on which they depend (no less, no more).
- Protect against multiple inclusion (include guards or **#pragma once**).

An ADT is a data type which:

- exports a name (type);
- defines the domain of possible values;
- establishes an interface to work with objects of this type (operations);
- restricts the access to the object components through the operations defined in its interface;
- hides the implementation.

Any program entity that satisfies the requirements from the ADT definition is considered to be an implementation of the ADT.

ADT implementation in C/C++:

- interface - header file (.h);
- implementation - source file (.c/.cpp).

# ADT Dynamic Array

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

## Requirement

Create a dynamic array, having a length that can be modified and allowing the insertion and deletion of elements of type *Planet*. Each *Planet* has:

- a name
- a type (Neptune-like, gas giant, terrestrial, super-Earth, unknown)
- the distance to the Earth (measured in light-years)

## DEMO

Dynamic array. (*DynamicArray.h*, *DynamicArray.c*).

# Summary I

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

## Memory management

- Memory allocation can be made statically (compile time) or dynamically (run time).
- We are responsible for dynamically allocated memory.
- We must pay attention to memory errors (memory leaks, repeated frees, dangling pointers, accessing memory at the NULL location).

# Summary II

Object-  
Oriented  
Programming

Iuliana  
Bocicor

Pointers

Memory  
management

Modular  
programming  
in C/C++

Abstract Data  
Types - ADT

Summary

## Modular programming

- Separate the interface from the implementation.
- Header (.h) files contain function declarations (define the "what").
- .c/cpp files contain function implementations (define the "how").