Chapter 8: Pointers

How can a program access or modify a variable indirectly? Is it possible for two variables to refer to the same memory location? What is the difference between a variable and its address in memory? How are arrays passed to functions in C? Why do some programs crash when accessing memory incorrectly? How does dynamic memory allocation work? What problems may occur if memory is not managed properly?

This chapter answers these questions.

8.1 Introduction

In programming, memory management is a fundamental concept that directly affects program correctness, efficiency, and reliability.
In languages such as C, programmers have direct access to memory through a powerful mechanism called pointers.

8.2 Memory and addresses

The main memory (RAM) used by programs is organized into bytes, each identified by a sequential number known as an address. By convention, addresses are represented in hexadecimal notation and prefixed with 0x.

Example: Memory Representation (RAM)

Adress	Content
0x1001	
0x1002	
0x1003	

- Each row represents one byte in memory.
- The address identifies the byte location in RAM.
- The content is the value stored at that address.
- A variable may occupy several consecutive addresses depending on its type.

Declaration of a variable means given a name (identifier) to a memory location.
This memory location is defined by:

- Its position: The address of its first byte,
- Its size: the number of bytes it occupies.

Example: Variables x:Integer;

x← 200;

Adress	Content	Location (Variable id)
0x1000		
0x1001	200	X
0x1002		
0x1003		
0x1004		
0x1005		
0x1006		
0x1007		
0x1008		
0x1009		
0x100A		

8.3 Definition of a Pointer

A pointer is a variable that stores the address of another variable in memory.
Instead of manipulating data directly, pointers allow programs to access and modify data indirectly, offering great flexibility and performance.
Pointers are widely used to:

- Manipulate arrays and strings,
- Pass parameters efficiently to functions,
- Implement dynamic data structures (lists, trees, stacks),
- Manage dynamic memory allocation.

However, improper use of pointers may lead to serious errors such as memory leaks, segmentation faults, or unpredictable program behavior.

For this reason, understanding pointers is essential for mastering the C language.

8.3 Declaration of a Pointer

The declaration of a pointer in algorithmics (C language) is performed as follows:

Syntax

In Algorithmic:

Variables pointer_id : ^Type;

In C language:

```
Type *pointer_id;
```

Example:

Variables P : ^ integer;

```
int *P;
```

8.4 Manipulation of a Pointer

Pointer manipulation refers to the different operations that can be performed on variables of type pointer.
These operations allow a program to store addresses, access the values stored at those addresses, and dynamically manage memory.

Pointers are widely used in data structures such as linked lists, trees, and dynamic arrays, where memory must be allocated and accessed through addresses.

The main operations involved in manipulating a pointer include:

- Initialization of the pointer.
- Assignment of an address to the pointer.
- Accessing the value pointed to by the pointer (dereferencing).
- Dynamic memory allocation.
- Memory deallocation.

8.4.1 Initialization

Initialization consists of giving an initial value to a pointer variable.

There are three ways to initialize a pointer variable:

- Initialization with the constant NIL (the pointer points to nothing).
- Initialization with the address of an existing variable (address of a variable or the value of another pointer).
- Initialization by dynamic memory allocation (allocate).

It should be noted that for a given pointer variable, only one of these three methods should be used at a time.

a. Initialization with NIL:

A pointer that is not initialized is called a wild pointer.
A wild pointer is a pointer variable that has been declared but has not been initialized to any valid memory address (such as NIL, the address of a variable, or dynamically allocated memory).

Because it contains a random or undefined address, using it may lead to unpredictable behavior or program crashes.
If we want the pointer to point to nothing, it must be initialized to NIL (Not Identified Link, NULL in C).

Example:

```
Variables   P : ^ integer;
    P← NIL;
```

b. Initialization with another address:

This method consists of initializing a pointer with the address of an existing variable or with the value of another pointer.
In this case, the pointer will point to an already existing memory location.

Principle

The pointer receives the address of a variable, so it becomes possible to access or modify the value stored in that variable through the pointer.

```
Variables   P : ^ integer;
            a:integer;
            T:^integer;
    a←15 ;
    P← &a;
    T←P ;
```

8.4.2 Dereferencing

Dereferencing a pointer means accessing the value stored at the memory address contained in the pointer.

In our pseudo-code notation, dereferencing is done using the symbol ^.

If P is a pointer, then:

P → contains an address
P^ → represents the value stored at that address

```
Variables   P : ^ integer;
            a:integer;

Begin
    a←15 ;
    P← &a;
    Write(P^);
    P^←100;
    Write(P^);
```

Notes:

- A wild pointer is a pointer that points anywhere in memory. If this pointer is dereferenced and assigned a new value, it may overwrite an arbitrary memory location, which can cause the program to crash.

- A pointer points to the variable whose address it contains.

- A pointer is associated with a specific variable type that it can point to. For example, a pointer to an integer can only point to integer variables.

8.4.3 Dynamic Memory Allocation

The **RAM of a computer** is composed of several types of memory, mainly:

- **Static memory**: a memory area where data having the same lifetime as the program are stored (global variables).
- **Automatic memory**: a memory area called the execution stack, where parameters, return addresses and values, and local variables of functions are stored. This memory is managed automatically by the compiler (allocation and deallocation).

- **Dynamic memory**: a memory area called the heap, in which the programmer can explicitly allocate memory. The programmer must also explicitly free this memory.

Dynamic memory allocation consists of reserving a memory space during program execution.

Unlike static variables whose memory is reserved when the program starts, dynamically allocated memory is created only when it is needed.

This technique is widely used in data structures such as linked lists, stacks, queues, and trees, where the number of elements is not known in advance.

In algorithmics, dynamic memory allocation is performed using the procedure allocate.

Principle

When a memory space is allocated:

- A new memory block is reserved.
- The address of this block is stored in the pointer.

Variables
P : ^ integer;

Begin
allocate(P);
P^ ← 25;
End.

- *allocate(P)* reserves a memory space for an integer.
- The address of the allocated memory is stored in P.
- P^ ← 25 stores the value 25 in that memory location.

Important Notes

After using dynamically allocated memory, it should be released to avoid memory waste. This is done using the Release operation: Release(P);

This instruction releases the memory space previously allocated to P.

After you release a pointer with free() in C, it's a good practice to set the pointer to NULL.

Why?

- After free(p), the pointer still holds the old address, but that memory is no longer valid.
- If you accidentally dereference it, you get a dangling pointer, which can crash your program.
- Assigning NULL ensures the pointer points to nothing, making it safe to check before use.

Warning: What You Must Not Do with Pointers

In programing, when working with pointers and dynamic memory, the following actions are forbidden:

1. **Do not free a pointer that is NULL (NIL).**
    - Freeing NULL may be harmless in some systems, but it is not good practice and can lead to confusion.
2. **Do not free memory that has already been freed.**
    - Freeing the same memory twice causes undefined behavior and can crash your program.
3. **Do not access a memory area that has been freed.**
    - After free(p), the memory is no longer valid. Dereferencing it leads to a dangling pointer error.
4. **Do not free memory that was allocated automatically by the compiler** (like local variables).
    - Only memory allocated with malloc, calloc, or realloc can be freed.
    - Attempting to free stack memory (automatic variables) causes undefined behavior.