Ministry of Higher Education and Scientific Research
Djilali BOUNAAMA University - Khemis Miliana(UDBKM)
Faculty of Science and Technology
Department of Mathematics and Computer Science

Chapter 3

Linked Lists

MI-L1-UEF121 : Algorithms and Data Structures II

Noureddine AZZOUZA

n.azzouza@univ-dbkm.dz

Course Topics

1. Memory Allocation
2. Pointers
3. Dynamic Allocation and Static Allocation
4. Linked Lists
   - 4.1 Definition
   - 4.2 LL Model
   - 1.3 LL Operations
   - 1.3 LL Variations

Memory Allocation

Introduction

- ✓ The memory space occupied by a dynamic data structure is variable.
- ✓ This is interesting for representing sets of variable sizes.
- ✓ We can therefore enlarge or shrink the size of the set during the execution of the program.
- ✓ Some problems require the management of a dynamic set.

Concept of memory allocation

- ✓ **Memory (Main Memory)** is made up of numbered cells. Each cell can store one byte (8 bits).

- ✓ A **variable** is a contiguous area in **MM** (a cell or a set of cells that follow one another).

  - ➢ Its **size** (in number of cells) depends on the type of the variable (eg: an integer occupies 4 cells, a real number occupies 8 cells, etc.).

  - ➢ The **address** of a variable is the number of its first cell.

Memory Allocation

Concept of memory allocation

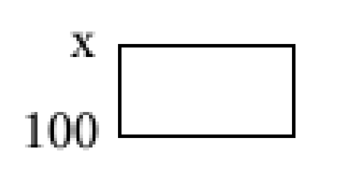
PASCAL

```
VAR x: integer;
```

C

```
int x;
```

✓ **x** is the **name** given to reference the memory location associated with the variable (the cell at address 100)



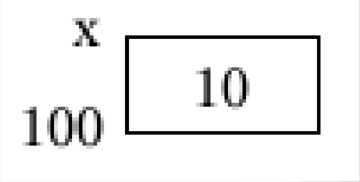
PASCAL

```
x := 10;
```

C

```
x = 10;
```

✓ when we assign a value (eg 10) to x, we then say that the content of address 100 is 10



Pointers


$password;
$database;
$charset;
$link = null;
function Connect()
self::$link = mysql_connect(self::
if (!self::$link)
throw new MysqlException("

Pointers : Declaration

- ✓ **a Pointer**: is a variable that can contain variable addresses.
- ✓ we can declare a pointer to an integer as follows:

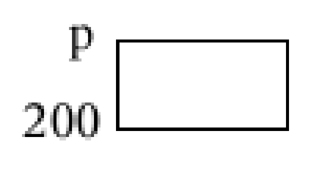
PASCAL

```
VAR p: ^integer;
```

C

```
int *p;
```

- ✓ according to this declaration, we can assign to the variable p the address of a variable of type “integer”.



- ✓ In C, the expression **&v** returns the address of the variable v. Some Pascal compilers (non-standard) offer such a mechanism (**Addr(v)** in turbo-pascal returns the address of the variable v)

Pointers

Pointers : Address of a variable

- ✓ so if we write for example:

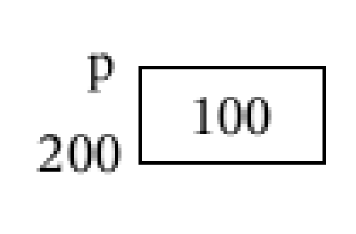
PASCAL

$p := Addr(x);$

C

$p = \&x;$

- ✓ we will have in p the address of x:



- ✓ we then say that p **points to** x.

Pointers : Indirect modification

✓ we can modify the value of x indirectly (without using x):

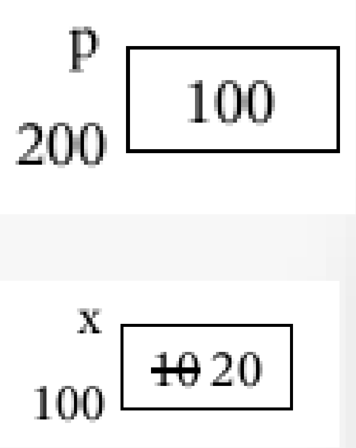
PASCAL

```
p^ := 20;
```

C

```
*p = 20;
```

✓ the value of x will then be modified (by an indirect assignment)



Dynamic Allocation and Static Allocation

Types of variable allocation

- ✓ Allocation of variables means creation of variables. Therefore reservation of memory space by associating with each variable the address of an empty area in memory.

Static Allocation

- ✓ Space allocation is done at the start of a treatment.
- ✓ managed automatically by the system
- ✓ the space is known at compile time.
- ✓ declared variables represent statically allocated variables

Dynamic Allocation

- ✓ Space is allocated as the program is executed.
- ✓ Managed manually by the programmer
- ✓ the space is unknown at compile time.
- ✓ The user must have both operations: allocation and release of space.

Dynamic Allocation

PASCAL

Allocation

new (p) : this procedure allocates a new variable and assigns its address to the variable p.

Release

dispose(p) : destroys the variable pointed to by p.

C

Allocation

malloc(nb_bytes): function which allocates a memory area of size nb_bytes and returns its address as result.

Release

free(p) : destroys the variable pointed to by p.

Dynamic Allocation

PASCAL

```
var
        p : ^char; { static allocation of a variable ( p ) of "pointer" type}
begin
        new(p);  { dynamic allocation of a character variable}
        p^ := 'A'; { indirect use of the dynamic variable}
        dispose(p); { destruction of dynamic variable}
end.
```

Dynamic Allocation

C

```
int main()
{
    char *p;                  /* static allocation of a variable ( p ) of "pointer" type*/
    p = malloc( sizeof(char) ); /* dynamic allocation of a variable of the same
    size as a character: sizeof( type ) returns the number of bytes needed to
    represent a variable of this type */
    *p = 'A';                 /* indirect use of the dynamic variable */
    free(p);                  /* destruction of dynamic variable*/
    return 0;
}
```

Notes

- ✓ **Null Pointer**

PASCAL	C
$p := NIL;$	$p = \mathrm{NULL};$

- ✓ The pointer constants NIL (in Pascal) or NULL or 0 (in C) indicate the absence of an address. So, for example in Pascal, the assignment p := NIL means that p does not point to any variable.

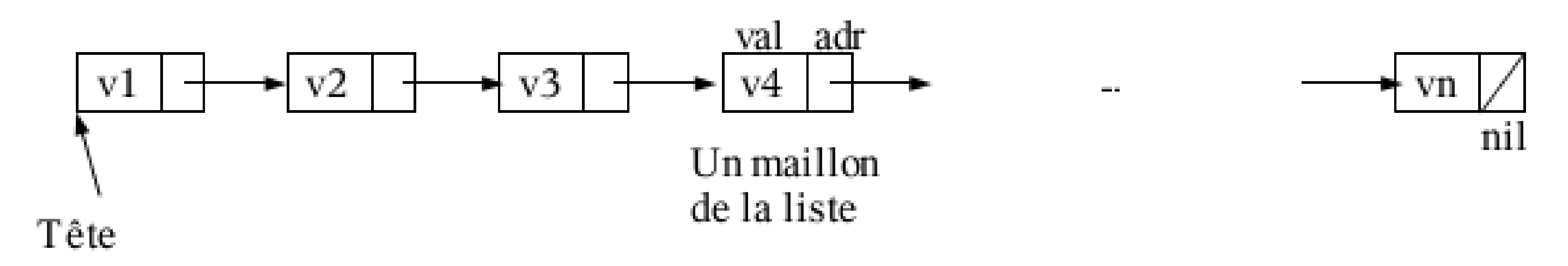
- ✓ You should never use indirection (^ in Pascal or * in C) with a pointer that does not contain the address of a variable, there will then be a segmentation fault.

Linked Lists

Definition

✓ A Linear Linked List is a data structure (most often dynamic) for representing a set of values.

✓ These values are chained together forming a sequence:



Properties

- ✓ Each value v of the set is stored in a node
- ✓ A node is a structure with 2 fields:
  - ➢ **val:** of any type,
  - ➢ **adr:** pointer to the next node
- ✓ In the **last node** of the list, the **adr** field contains the **NIL** pointer (indicating by convention the end of the list).
- ✓ The address of the **1st node** (**the head of the list**) is important. It must always be saved in a variable to be able to manipulate the list.
- ✓ If the list is empty (contains no links), the head must then be positioned at NIL.

LL Model : Node Structure

✓ In algorithmic language, we define the type of a node as follows

```
Type    node = Record
                  Val : Typeqq /*any type*/
                  Adr : Pointer(node)
        EndNode
```

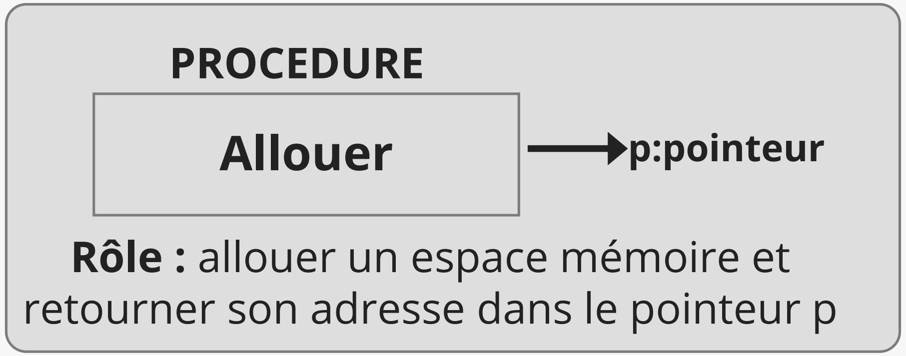
LL Model : Abstract Data Types

✓ In order to develop algorithms on LLs, we build an abstract machine (**Abstract Data Types - TAD**) with the following operations :

- **Allocate(P)**: allocation of a space of size specified by the type of P. The address of this space is rendered in the Pointer type variable P.
- **Release (P)**: release of the space pointed by P.
- **Value (P)**: consultation of the "Value" field of the node pointed to by P.
- **Next (P)**: consultation of the "Address" field of the node pointed to by P.
- **Aff_Adr(P, Q)**: in the "Address" field of the node pointed to by P, we put the address Q (Adress saved in the pointer Q).
- **Aff_Val(P,Val)**: in the "Value" field of the node pointed to by P, we store the value Val.

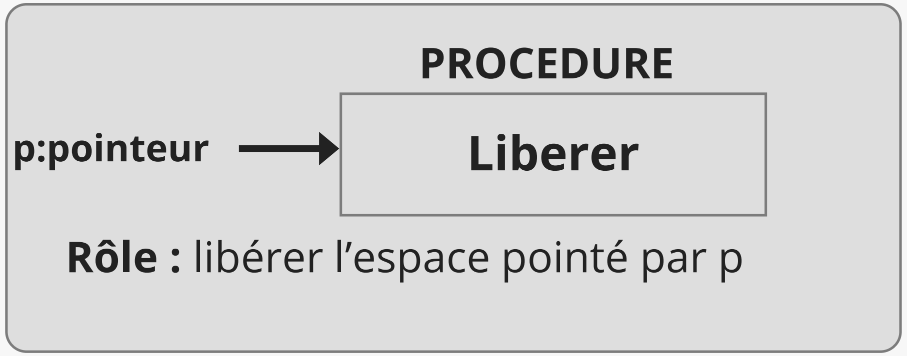
LL Model : Implémentation de la TAD

Allouer(P)




```
Procédure Allouer (VAR p: pointeur(maillon))
Début
    new(p); //ou malloc() en C
Fin;
```

Liberer(P)




```
Procédure  Liberer (p: pointeur(maillon))
Début
    dispose(p); //ou free(p) en C
Fin;
```

LL Model : Implémentation de la TAD

Valeur(P)

FONCTION

p:pointeur → **Valeur** → Type qlq

Rôle : retourner la valeur du champ « Val » du maillon pointé par P

Fonction *Valeur (p: pointeur(maillon)): type qlq*

Début

$\quad Valeur \leftarrow p\hat{}.val;$

Fin;

Suivant(P)

FONCTION

p:pointeur → **Suivant** → p:pointeur

Rôle : retourner la valeur du champ « Adr » du maillon pointé par P

Fonction *Suivant (p: pointeur(maillon))*

Début

$\quad Suivant \leftarrow p\hat{}.adr;$

Fin;

LL Model : Implémentation de la TAD

Aff_Val(P,V)

PROCEDURE

V: type qlq → **Aff_Val** → p:pointeur

Rôle : dans le champ « Val » du maillon pointé par P, on range la valeur « V »

Procédure *Aff_ Val (V:typeqlq;* ***VAR*** *p: pointeur(maillon))*

Début

$p\hat{}.val \leftarrow V;$

Fin;

Aff_Adr(P, Q)

PROCEDURE

q:pointeur → **Aff_Adr** → p:pointeur

Rôle : dans le champ « Adr » du maillon pointé par P, on range la valeur « Q »

Procédure *Aff_ Adr (Q:pointeur(maillon);* ***VAR*** *P: pointeur(maillon))*

Début

$p\hat{}.adr \leftarrow Q;$

Fin;

Example : Create a list of N elements

```
Algorithm  CreateList;
    Type  node = Record
              Val:integer;
              Adr:pointer(node)
          EndNode;
    Var   Head, P, Q : pointer(node);
          i, N, V : integer ;
Begin
    Head := Null;
    P  := Null;
    Write('Gide the number of elements :');
    Read (N);
```

```
For i :=1 to N Do
    Read(Val) ;
    Allocate(Q) ;
    Aff_val(Q, val) ;
    Aff_adr(Q, NULL) ;
    if (Head <> NULL) then
        Aff_adr(P, Q)
    else
        Head := Q
    endif;
    P := Q;
  endfor;
```

Example : Browse and view list items

```
P := Head ;
While (P <> NULL) Do
    Write(Value(P)) ;
    P := Next(P) ;
endWhile;

End.
```