Linked Lists

D.E ZEGOUR

École Supérieure d'Informatique

ESI

Linked lists

Introduction example

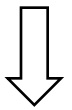
Find all prime numbers from 1 to n and store them in memory.

n : data to read

Problem : choice of the data structure

If Array : not possible to define its size

If Array : not possible to precisely define its size even if n is known



Space allocation must be dynamic

Linked lists

Concept of Static and Dynamic Allocation

Static allocation

Space is entirely allocated at the beginning of a process

In technical terms, we describe this as the space being known at compile time

It ‘s therefore an array

Dynamic allocation

Space is allocated as the program executes

To perform this type of allocation, the user must have access to both space allocation and space deallocation operations.

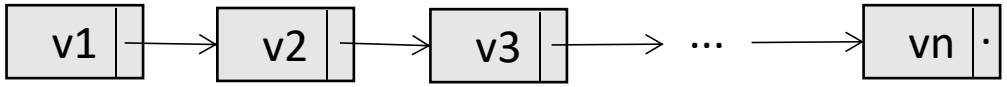
If the programming language provides these capabilities, they can be used directly

Otherwise, simulate them, meaning managing the space manually within a large array.

Linked lists

Definition

A linked list (LL) is a collection of dynamically allocated nodes (or cells) linked together



Head

An item of an LL is always a structure with two fields :
- Value field : holding the information
- Address field : giving the address of the next cell

An LL is characterized by the address of its first element.

Each node is associated with an address

Nil represents an address that does not point to any node

This introduces in the algorithmic language a new variable type: the POINTER type

Linked lists

Abstract Machine (AMLL)

ALLOCATE _CELL(P) :

Create a cell and return its address in P.

FREE (P) :

Free the node of address P.

NEXT (P) :

Access to the Address field of the node referenced by P.

CELL_VALUE (P) :

Access to the Value field of the node referenced by P.

ASS_ADR (P, Q) :

Assign to the Address field of the node referenced by P, the address Q.

ASS_VAL(P, Val) :

Assign to the Value field of the node referenced by P, the value Val.

Linked lists

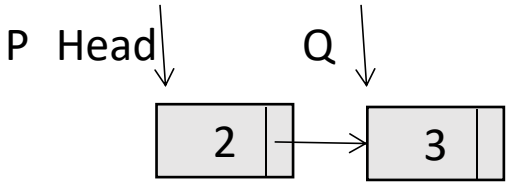
Solution to introduction problem

We use the following two facts next:

1. A number is prime only if it is not divisible by the prime numbers that precede it.

2. All prime numbers are of the form 6m+1 or 6m-1.

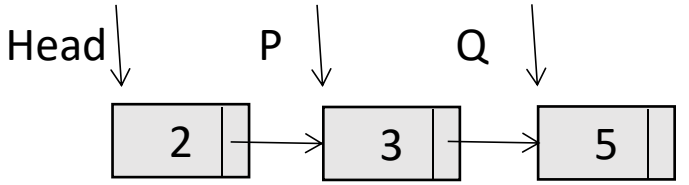
Linked lists



Solution to introduction problem

```
READ ( N ) ;
ALLOCATE_CELL ( P ) ;
ASS_VAL ( P , 2 ) ;
WRITE ( 2 , 3 ) ;
Head := P ;
ALLOCATE_CELL ( Q ) ;
ASS_VAL ( Q , 3 ) ;
WRITE ( 3 ) ;
ASS_ADR ( Q , NULL ) ;
ASS_ADR ( P , Q ) ;
```

Linked lists



Solution to introduction problem

```
READ ( N ) ;
ALLOCATE_CELL ( P ) ;
ASS_VAL ( P , 2 ) ;
WRITE ( 2 , 3 ) ;
Head := P ;
ALLOCATE_CELL ( Q ) ;
ASS_VAL ( Q , 3 ) ;
WRITE ( 3 ) ;
ASS_ADR ( Q , NULL ) ;
ASS_ADR ( P , Q ) ;
```

```
M := 1 ;      Continue := TRUE ;      Aig := TRUE ;
WHILE Continue :
   CALL Gen_Number ( M , Aig , Number ) ;
   Aig := NOT Aig ;
   IF Aig :
      M := M + 1
   ENDIF ;
   IF Number <= N :
      IF Prime ( Head , Number ) :
         WRITE ( Number ) ;     ALLOCATE_CELL ( Q ) ;
         ASS_VAL ( Q , Number ) ;     ASS_ADR ( Q , NULL ) ;
         ASS_ADR ( P , Q ) ;
         P := Q
      ENDIF
   ELSE
      Continue := FALSE
   ENDIF
ENDWHILE ;
```

Linked lists

Solution to introduction problem

Gen_nombre(M, Aig, Number)

```
If Aig : Number := 6M - 1
Else Numbre := 6M + 1 Endif
```

Divisible (A, B)

```
Q := A / B {Integer division }
Divisible := Q.B = A
```

Prime (L, N)

```
P := L ;
Found := FALSE ;
WHILE ( P <> NULL ) AND NOT Found :
   IF Divisible ( Number , CELL_VALUE ( P ) ) :
      Found := TRUE
   ELSE
      P := NEXT ( P )
   ENDIF
ENDWHILE ;
Prime := NOT Found
```

Linked lists

Algorithms on linked lists

Just like with arrays, algorithms on LLs can be classified as follows:

1. Traversal: access by value, access by position

2. Updates: insertion, deletion

3. Algorithms involving multiple LLs: merging, interleaving, splitting, etc.

4. Sorting

Linked lists

Special Linked Lists

Doubly linked list (DLL)
It is an LL that can be traversed in both directions

AMDLL = AMLL - {ASS_ADR} + { ASS_R_ADR, ASS_L_ADR, PREVIOUS }

The abstract machine on LL is extended by the following operations:

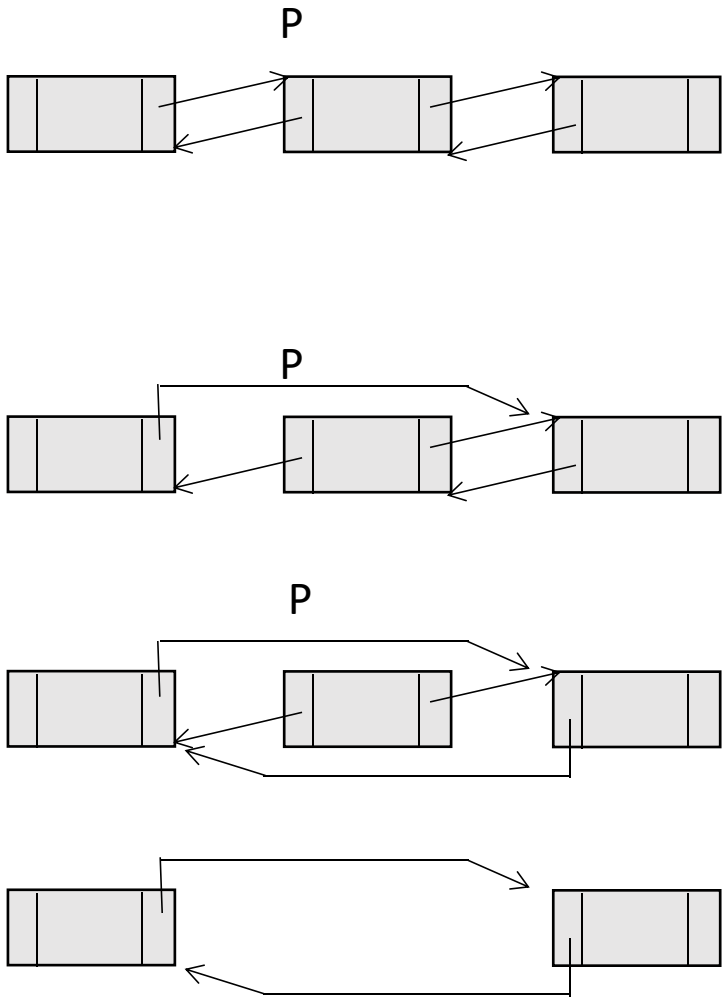
PREVIOUS (P) : Access to the 'Left address' field of the node referenced by P.
ASS_R_ADR (P, Q) : Assign to the 'Right address' field of the node referenced by P, the address Q.
ASS_L_ADR (P, Q) : Assign to the 'Left address' field of the node referenced by P, the address Q.

Linked lists

Special Linked Lists

Deleting the item pointed by P in a doubly linked list L

```
IF P # NULL :
      IF PREVIOUS ( P ) # NULL :
            ASS_R_ADR ( PREVIOUS ( P ) , NEXT ( P ) )
      ELSE
            L := Next (P)
      ENDIF
      IF NEXT ( P ) # NULL :
            ASS_L_ADR ( NEXT ( P ) , PREVIOUS ( P ) )
      ENDIF
      FREE ( P ) ;
FSI
```



Linked lists

Special Linked Lists

Circular Linked List (CLL)

It is an LL in which the last element points to the first. It is defined by the address of any element

AMLLC = AMLL

Linked lists

Special Linked Lists

Circular Doubly Linked List (CDLL)

It is a two-way CLL in which the last (first) element points to the first (last)

AMCDLL = AMDLL.

Linked lists

Implementation : Dynamic (C)

```
#include <stdio.h>
#include <stdlib.h>

/** -Implementation- **\: LIST Of INTEGERS**/
/** Linked lists **/

typedef int Typeelem_Li   ;
typedef struct Cell_Li * Pointer_Li ;
struct Cell_Li
 {
   Typeelem_Li  Val ;
   Pointer_Li Next ;
 } ;
Pointer_Li Allocate_cell_Li (Pointer_Li *P)
 {
   *P = (struct Cell_Li *) malloc( sizeof( struct Cell_Li)) ;
   (*P)->Next = NULL;
 }
void Ass_val_Li(Pointer_Li P, Typeelem_Li Val)
 {
   P->Val = Val ;
 }
```

```
void Ass_adr_Li( Pointer_Li P,  Pointer_Li Q)
 {
   P->Next = Q ;
 }

Pointer_Li Next_Li(  Pointer_Li P)
 { return( P->Next ) ;  }

Typeelem_Li Cell_value_Li( Pointer_Li P)
 { return( P->Val) ; }
void Free_Li ( Pointer_Li P)
 { free (P);}
/** Variables of main program **/
Pointer_Li L=NULL;

int main(int argc, char *argv[])
 {
   system("PAUSE");
   return 0;
}
```

Linked lists

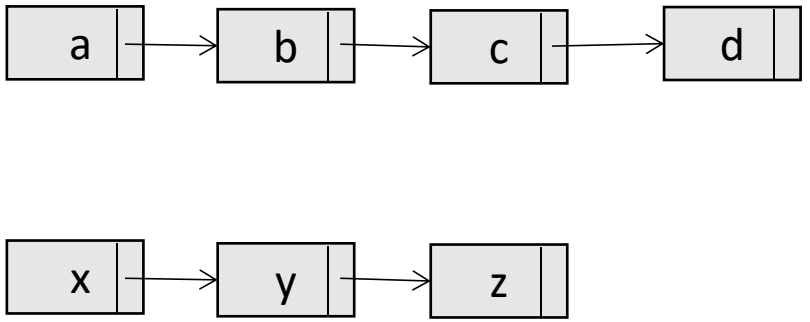
Implementation : Static (C)

The array is a set of triples (Element, Next, Occupied)

The 'Element' field holds the stored value.

The 'Next' field contains the address of the next cell.

The 'Occupied' field indicates the availability of the cell. It is necessary for 'Allocate' and 'Free' operations.



Static (or global) Array

0	y	6	V
1	d	-1	V
2	b	5	V
→ 3	a	2	V
→ 4	x	0	V
5	c	1	V
6	z	-1	V
			F
			F
Max -1			

Linked lists

Implementation : Static (C)

Linked lists can be represented in a single array

An initialization phase is mandatory before using this array. It consists of initializing the 'Occupied' field to false.

A linked list is defined by the index of its first element.

```
#define Max 100
#define True 1
#define False 0
#define Nil -1

typedef int Bool;
typedef int Anytype;
struct Typelist
 {
  Anytype Element ;
   int Next;
   Bool Occupied;
 } ;
struct Typelist List[ Max ];
```

Linked lists

Implementation : Static (C)

```
void Allocate ( int *I )
 {
   Bool Found;
   *I = 0;
   Found = False;
   while ( *I < Max && !Found)
     if ( List[*I].Occupied )
       *I++  ;
     else
       Found= True;
     if  ( !Found)  *I = -1;

 }
void Free ( int I )
 {
   List[I].Occupied = False ;
 }
Anytype Value ( int I )
 {
   return( List[I].Element );
 }
```

```
int Next ( int I )
 {
   return ( List[I].Next ) ;
 }
void Ass_val ( int I, Anytype Val)
 {
   List[I].Element = Val;
 }

void Ass_adr ( int I, int J)
 {
   List[I].Next = J;
 }
int main(int argc, char *argv[])
 {
   system("PAUSE");
   return 0;
 }
```

Linked lists

Implementation : Dynamic (Pascal)

```
PROGRAM My_program;
{ -Implementation- : LIST Of INTEGERS}
{ Linked lists }
TYPE
  Typeelem_LI = INTEGER;
  Pointer_LI = ^Cell_LI; { type of field 'Address' }
  Cell_LI = RECORD
     Val : Typeelem_LI;
     Next: Pointer_LI
  END;

PROCEDURE Allocate_cell_LI ( VAR P : Pointer_LI ) ;
  BEGIN  NEW(P)  END;

PROCEDURE Free_LI ( P : Pointer_LI ) ;
  BEGIN  DISPOSE(P)  END;

PROCEDURE Ass_val_LI(P : Pointer_LI; Val : Typeelem_LI );
  BEGIN
    P^.Val := Val
  END;
```

```
FUNCTION Cell_value_LI (P : Pointer_LI) : Typeelem_LI;
  BEGIN   Cell_value_LI := P^.Val  END;

FUNCTION Next_LI( P : Pointer_LI) : Pointer_LI;
  BEGIN  Next_LI := P^.Next END;

PROCEDURE Ass_adr_LI( P, Q : Pointer_LI ) ;
  BEGIN P^.Next:= Q    END;

{ Declaration part of variables }
VAR
  L : Pointer_LI;

 { Body of main program }
 BEGIN
   READLN;
 END.
```