University Echahid Hamma Lakhdar of El-oued
Department of Computer Science
Level: 2nd year LMD Computer Science
Module: Algorithms and Data Structures
(Pointers)

The syntax to declare a pointer is as follows.

```
type *PointerName;
```

For example, if we want to create a pointer to int (that is, a pointer that can store the address of an int object) and call it "ptr", we must write this.

```
int *ptr;
int * ptr;
int* ptr;
```

Initialization
A pointer, like a variable, has no default value, so it's important to initialize it to avoid possible problems. To do this, you need to use the address-of (or referencing) operator &, which returns the address of an object. This operator is placed in front of the object whose address you want to obtain. For example:

```
int a = 10;
int *p;
p = a;
```

```
include < stdio.h >
int main(void)
{
int a = 10;
int *p;
p = a;
printf("a = % d", a);
printf("*p = %d", *p);
printf("&a = %d", &a);
printf("p = %d", p);
printf("&p = %d", &p);
return 0;
}
```

Results

```
**********************************************************
a = 10
*p = 10
&a = 6487580
p = 6487580
&p = 6487568
**********************************************************
```

NB:
Be careful not to mix different pointer types! A pointer to int is not the same as a pointer to long or double. Likewise, assign the address of an object only to a pointer of the same type.

```
int a;
double b;
int *p = &b; /* Wrong */
int *q = &a; /* Correct */
double *r = p; /* Wrong */
```

The NULL constant
To make source code clearer, there is a constant defined in the ¡stddef.h¿ header: NULL. It can be used anywhere a null pointer is expected.

```
int *p = NULL; /* A null pointer */
```

Pointer to Pointer:

```
include < stdio.h >
int main(void)

int a = 10;
int *pa = &a;
int **pp = &pa;
printf("value of a before modifying pointer pa = %d", a);
*pa=20;
printf("a = %d", a);
printf("*pa = %d", *pa);
printf("pa = %d", pa);
printf("**pp = %d", **pp);
printf("*pp = %d", *pp);
printf("pp = %d", pp);
return 0;
```

Results

****

value of "a" before modifying pointer "pa": a = 10
value of "a" after modifying pointer "pa": a = 20
*pa = 20
pa = 6487572
**pp = 20
*pp = 6487572
pp = 6487560

****

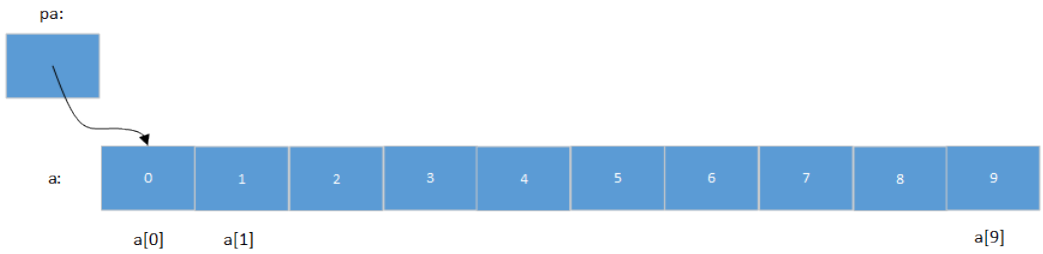
Pointers and Arrays

- Every operation with array indices can also be expressed by pointers.
- If we assign the name of an array to a pointer, it points to the address of the first element of the array (&array and array mean the same address of the first element) .

  int a[10];
  int * Pa;
  Pa = a// equivalent to Pa=& a[0].

- If P points to some element of the array, then P+i points to the following element.

  P + i points to the i-th element to the right of *P.
  P - i points to the i-th element to the left of *P.



If P=Tab, then:

```
*(P+1) means Tab[1];
*(P+2) means Tab[2];
.....
*(P+i) means Tab[i];
```

- Incrementing and decrementing a pointer.

```
If P points to Tab[i], then after: P++; P points to Tab[i+1].
P+=n; P points to Tab[i+n].
P−−; P points to Tab[i-1].
P− =n; P points to Tab[i-n].
```

Note: If P points to Tab[i], then P+i does not take the value of the i-th byte after P but rather the address of the i-th component after P — it depends on the pointer type, as each type addresses a fixed number of bytes per element.

If T is an array of floats and P is a pointer of the same type:

float T[20], x;
float *p;
after the instructions:
P=T;
x=*(P+5);

x contains the value of the sixth element of T, T[5].
Since a float uses 4 bytes, the compiler gets the address P + 5 by adding 5 * 4 = 20 bytes to P's address.

Subtraction and Comparison of Two Pointers:

- Subtracting two pointers:
  If P1 and P2 are two pointers pointing to the same array: - (P1-P2) gives the number of elements between P1 and P2 in the array, not the number of bytes.

The result of (P1-P2) is:
- Negative if P1 precedes P2.
- Zero if P1=P2.
- Positive if P2 precedes P1.
- Undefined if P1 and P2 are not in the same array.

- Comparing two pointers:
  We can compare two pointers with: >, <, <=, >=, ==, ! =.
  If both pointers are in the same array, the comparison is between indices; otherwise, the comparison is between their addresses in memory.

Summary

Suppose Tab is an array of any type:
- Tab defines the address of Tab[0];
- Tab+i defines the address of Tab[i];
- *(Tab+i) defines the content of Tab[i];

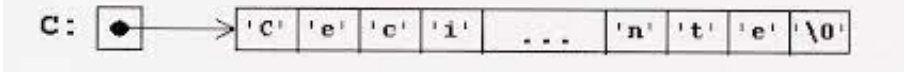
If P=Tab then:
- P points to Tab[0];
- P+i points to Tab[i];
- *(P+i) defines the content of Tab[i];

Pointers and Strings

- Everything said about pointers and arrays remains true for pointers and strings.
- Additionally, a pointer to a char variable can also hold the address of a constant string and can even be initialized with such an address.

Exemple

```
char *C;
C = "Ceci est une chaîne de caractères constante";
```




```
char *ch="Good evening!";
```

```
char *ch1="hello";
char *ch2="hello everyone";
ch1=ch2;// ch1 now points to the second string.
```

Note: A char pointer can point to strings of any length.

Pointers and Two-Dimensional Arrays

Let Mat be an integer array such that:

```
int Mat[4][6];
... If a pointer P=Mat, then P points to Mat[0] (the first row).
If P=Mat+i, then P points to the i-th row of the matrix.
The value (*(Mat+2))[3]=16.
```

Problem: how to access an element of a matrix using only pointers.

Solution: if Mat is a matrix and P is a pointer to it, i.e. P=Mat, then to access an element we must convert P to int* like this:

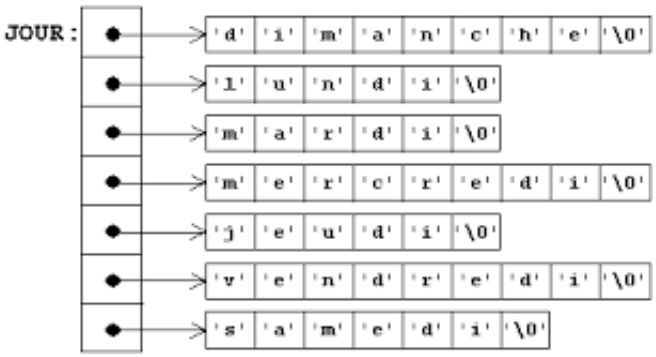
```
... int *P=Mat;
int *P1;
//force type conversion.
P1=(int*)P;
or
P1=(int*)Mat;
```

NB: P1 now points to Mat[0][0], and the matrix can be treated as a one-dimensional array of size 24.
Example: sum of matrix elements using pointers

```
... P=(int*)Mat;
sum=0;
for(i=0;i<24;i++)
{
sum+=*(P+i);
}
```

Array of Pointers
Example: int *Tab[10]; // Tab is an array of 10 pointers to int.

```
char *Month[ ]={"January", "February", ...};
... for(i=0;i<12;i++)
printf("%c", *Month[i]);
```



To display the content of the cell, you need to use a pointer to the array of pointers
...

Result:

```
**********************************************************
eloued
alger
oran
media
**********************************************************
```

The Generic Pointer The type void * is a generic pointer type, i.e. able to point to any object type.

...

Summary

- A pointer is a variable whose content is an address;
- The address-of operator & retrieves a variable's address;
- A pointer of one type can only hold the address of an object of the same type;
- A null pointer contains an invalid address depending on your OS;
- A null pointer is obtained by converting zero to a pointer type or using the NULL macro constant.
- The indirection operator (*) accesses the object referenced by a pointer;
- When returning a pointer from a function, ensure the referenced object still exists;
- The void type allows building generic pointers;
- The format specifier %p can be used to display an address (converted to void * first).