Chapter 2 : Indexed variables

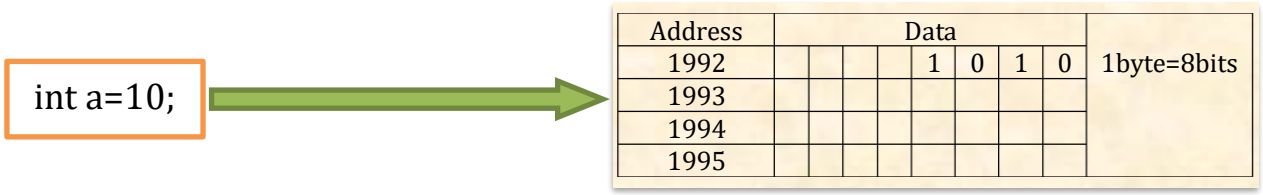
Part II : Pointers

C Memory address:

C memory address is the space required for storing a declared variable, or more appropriately, it is the location of bytes occupied by the variable data (value) in the computer.

For example the statement `int a=10 ;` leads to :

1- A storage size with 4 bytes is reserved in the memory.
2- The value 10 of **a** is converted to binary as $(10)_{10}$ =1010 and stored in the reserved memory of **a** as follows



In a C program, the address memory is frequently presented, particularly when using the **scanf()** function.

The statement `scanf(%d, &b);` means assigning a value to the address of **b**, which is referred to as **&b**.

It should be noted that the address operator '**&**' can be used to access the memory address of a variable.

Pinters in C :

Pointers are variables that store the address of another variable as their values. In c, pointers it can be created as follows :

- **Declaration** : Pointer is declared in this way

  pointer_data-type* pointer_name ;
  The only difference between pointer and variable declarations is the astrisic (*).
  Example :
  int* p ;
  This line means that a pointer is created but it points to a random adress memory. The declared pointer keeps pointing to an existing and unknown address in memory, and if it is not specified, a programming issue may arise. To this reason, a care should be done in pointers manipulation.

- **Initialization:** the pointer is initialized by using address operator '&' as

  int a=5, *pa ;
  pa=&a ;

  This indicates that the pointer **pa** has been assigned the address of **a**. Also, the stored value of **a** is accessed using the dereference operator ***.**

The following example will clarify everything that has been mentioned in this section.

Example :

Program	*Output*

#include <stdio.h>
 	
int main() {
  int a=5, *pa;
  pa=&a;
  printf("The value of a=%d\n",a);
  printf("The address of a: %p\n", &a);
  printf("The value of a=%d\n", *pa);
  printf("The address of a: %p", pa);
 	
  return 0;
}
	The value of a=5
	The address of a= 0x7ffe665a0c14
	The value of a=5
	The address of a= 0x7ffe665a0c14

From this example, the importance of pointers also becomes clear, as they play two roles. On the one hand, they store the address of the variable, and on the other hand, they can access the value of the variable they pointed to. Even beyond that, the value of the variable it can be changed through its pointer.

Example:

Program	*Output*

#include <stdio.h>
 	
int main() {
  int a=5;
  int* pa;
  pa=&a;
  *pa=10;
  printf("a=%d\n", a);
  return 0;
}
	a=10

Pointers and functions:

Pointers are useful tools to deal with functions. It can be pointed to the function itself or it can be used as a parameter to that function. In most cases pointers are used to pass addresses to function parameters and this technique is called **pass by reference.** This it can be explored in the programs as follows

Example 1:

Program	*Output*

#include <stdio.h>
void fg(int *q)
{
  printf("%p\n%d", q,*q);
}
int main() {
  int a=5, *p;
  p=&a;
  printf("%p\n",p);
  fg(p);
  return 0;
}
	0x7ffd26425cf4
	0x7ffd26425cf4
	5

Example 2:

Program	*Output*

#include <stdio.h>
void swap(int *p1, int *p2)
{
  int temp;
  temp=*p1;
  *p1=*p2;
  *p2=temp;
  printf("x=%d\ny=%d\n",*p1,*p2);
}
int main() {
  int x=5,y=10;
  printf("The value of x and y before swap\n");
  printf("x=%d\ny=%d\n", x, y);
  printf("The value of x and y after swap\n");
  swap(&x,&y);
  return 0;
}
	The value of x and y before swap
	x=5
	y=10
	The value of x and y after swap
	x=10
	y=5

Pointers and arrays :

The pointer of an array can be shown as a second array, and the elements of this array store the addresses of the first array elements to which it points.

Example :

Program	*Output*

#include <stdio.h>
 	
int main() {
  int i, j, a[4]={5,4,3,1};
  int* pa;
  pa=&a;
  printf("The address of a[] : %p\n\n", pa);
  printf("The addresses of array's elements:\n");
  for(i=0;i<4;i++)
  {
  printf("a[%d] : %p\n", i, pa+i);
  }
  printf("The values of array's elements:\n");
  for(j=0; j<4; j++)
  {
  printf("a[%d]=%d\n", j, *(pa+j));
  }
  return 0;
}
	The address of a[] : 0x7ffe14beea20
	
	The addresses of array's elements:
	
	a[0] : 0x7ffe14beea20
	a[1] : 0x7ffe14beea24
	a[2] : 0x7ffe14beea28
	a[3] : 0x7ffe14beea2c
	
	The values of array's elements:
	
	a[0]=5
	a[1]=4
	a[2]=3
	a[3]=1

From the previous example, it can be noted that:

- The address of array a[] corresponds to the address of a[0].
- The addresses are shifted by 4, and this is due to the fact that each element with data-type integer reserves 4 bytes in memory.
- The method to access array elements is done in the same way as
  - a[0]=*pa=5,
  - a[1]=*(pa+1)=4,
  - a[2]=*(pa+2)=3,
  - a[3]=*(pa+3)=1

Pointers can only be used for one element in the array.

```
int c[]={1, 2, 3}, *pc;
pc=&c[1];
```

In this case the order becomes important, so **pc+1** refers to the second element after c[1] and **pc-1** refers to the second element before c[1].

Example:

Program	*Output*

#include <stdio.h>
 	
int main() {
  int c[]={1, 2, 3}, *pc;
  pc=&c[1];
  printf("%d\n%d", *(pc-1), *(pc+1));
}
	1
	3

Example:

Here is a simple example of printing elements for every given array using a pointer, function, and an array.

Program	*Output*

##include <stdio.h>
void funcar(int *pa, int size)
{
int i;
for(i=1; i<size; i++)
{
printf("a[%d]=%d\n", i, *(pa+i));
}
}
int main() {
int b, a[]={4, 2, 3, 12, 51}, *p;
p=&a;
b=sizeof(a)/sizeof(a[0]);
funcar(p,b);
return 0;
}
	a[1]=2
	a[2]=3
	a[3]=12
	a[4]=51