Pointer

- ## First let's see how memory looks like:

Address	Content
x1000	a = 6
x1004	
x1008	b = 8
x100c	c = 10
x1010	
.	
.	
.	
XXXX	. . . . .

Memory

In the above figure we can see the variable a, b and c containing value 6, 8 and 10 is stored in memory address X1000, X1008 and X100c respectively.

So far we have learned that we can declare variable of certain types (int, float, char etc) and can do different operations on those variable to solve particular problems. But can we work directly with the memory address of those variables?

- ## What is pointer?

Pointer is a variable that can contain the memory address of another variable. A pointer can contain the:

- Memory Address of:
  - A primitive (int , char, float , double)
  - Array, String
  - Dynamically allocated Memory
  - Another pointer
  - Structure or Union
  - Function.
- A Null value.
- A garbage value (i.e. invalid memory address)

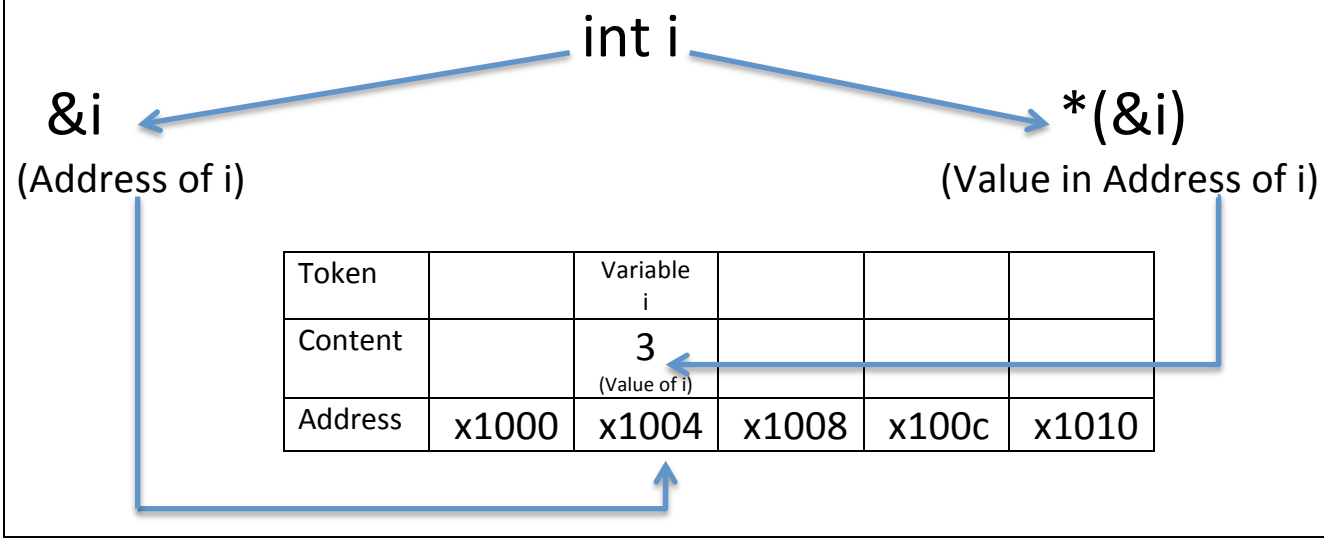
- **Operator used in pointer**

&	*
Address-of operator	**Dereferencing operator**

Consider the following code

```
#include<stdio.h>
  int main(){
        int i= 3;
        int *p;
        p = &i;                    // assigning the memory address of i to pointer p

        printf("%x\n", &i);        // &  gives the address of i variable
        printf("%d\n", *(&i));     // * dereferencing the  memory address of   variable i and
                                      prints the value stored in that address.
        printf("%x\n", p);         // p now gives the address of i variable
        printf("%x\n", *p);        // printing the content stored in address assigned to p.
}
```



- **Syntax for pointer**

datatype *identifier

Example:

```
int *pointer_for_integer;
float *pointer_for_float;
double *pointer_for_ double;
char * pointer_for_ char;
```

Try to understand the following codes:

Practice 1:

```
#include<stdio.h>
int main(){
        int a[5] = {1,3,5,7,9};
        int *p_integer;
        p_integer = &a;
        printf("Adrress=%x--Content=%d\n",&a, a[0]);
        printf("Adrress=%x--Content=%d\n",&a[1], a[1]);
        printf("Adrress=%x--Content=%d\n",&a[2], a[2]);
        printf("Adrress=%x--Content=%d\n",&a[3], a[3]);
        printf("Adrress=%x--Content=%d\n",&a[4], a[4]);
        printf("\nUsing pointer:\n");
        printf("Adrress=%x--Content=%d\n",p_integer+0, *(p_integer+0));
        printf("Adrress=%x--Content=%d\n",p_integer+1, *(p_integer+1));
        printf("Adrress=%x--Content=%d\n",p_integer+2, *(p_integer+2));
        printf("Adrress=%x--Content=%d\n",p_integer+3, *(p_integer+3));
        printf("Adrress=%x--Content=%d\n",p_integer+4, *(p_integer+4));
        return 0;
}
```

Practice 2:

```
#include<stdio.h>
 int main() {
         int a[5] = {1,3,5,7,9}, i;
         int *p_integer;
         p_integer = &a;
         printf("Adrress=%x--Content=%d\n",&a, a[0]);
         printf("Adrress=%x--Content=%d\n",&a[1], a[1]);
         printf("Adrress=%x--Content=%d\n",&a[2], a[2]);
         printf("Adrress=%x--Content=%d\n",&a[3], a[3]);
         printf("Adrress=%x--Content=%d\n",&a[4], a[4]);
         printf("\nUsing pointer:\n");
         printf("Adrress=%x--Content=%d\n",p_integer, *p_integer++);
         printf("Adrress=%x--Content=%d\n",p_integer, *p_integer++);
         printf("Adrress=%x--Content=%d\n",p_integer, *p_integer++);
         printf("Adrress=%x--Content=%d\n",p_integer, *p_integer++);
         printf("Adrress=%x--Content=%d\n",p_integer, *p_integer++);
```

```
        }
        return 0;
}
```

Try to answer:

What will happens if you use *++p_integer in the above code.

Practice 3:

```
#include<stdio.h>
 int main() {
        int a[5] = {1,3,5,7,9}, i;
        int *p_integer;
        p_integer = &a;
        printf("Adrress=%x--Content=%d\n",&a, a[0]);
        printf("Adrress=%x--Content=%d\n",&a[1], a[1]);
        printf("Adrress=%x--Content=%d\n",&a[2], a[2]);
        printf("Adrress=%x--Content=%d\n",&a[3], a[3]);
        printf("Adrress=%x--Content=%d\n",&a[4], a[4]);
        printf("\nUsing pointer:\n");
        for(i=0; i<5; i++){
        printf("Adrress=%x--Content=%d\n",p_integer, *p_integer+i);
        }
        return 0;
}
```

• Dynamic Memory Allocation:

Dynamic memory allocation refers to manual memory management. This allows you to obtain more memory when required and release it when not necessary.

There are 4 library function defined under **<stdlib.h>** for dynamic memory allocation.

FUNCTION	USE OF FUNCTION
malloc()	Allocates requested size of bytes and returns a pointer first byte of allocated space
calloc()	Allocates space for an array elements, initializes to zero and then

	returns a pointer to memory
free()	deallocate the previously allocated space
realloc()	Change the size of previously allocated space

- **malloc():**

  The function malloc() reserves a single block of memory of specified size and return a pointer of type void which can be casted into pointer of any form.

  Syntax of malloc():

```
pointer = (cast-type*) malloc(element-size)
```

- **calloc():**

  The only difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks (n blocks) of memory each of same size and sets all bytes to zero.

  Syntax of malloc():

```
pointer = (cast-type*)calloc(n, element-size);
```

- **free():**

  Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on its own. It must be explicitly freed using free() to release the space.

  Syntax of free():

```
free(pointer);
```

- **realloc():**

  If the previously allocated memory is insufficient or more than required, we can change the previously allocated memory size using realloc().

  Syntax of realloc():

```
previous_pointer = realloc(previous_pointer, newsize);
```

Practice 3:

Write a C program to find sum of n elements entered by user while dynamically allocating the memory using malloc() function.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int num, i, *ptr, sum = 0;
printf("Enter number of elements: ");
scanf("%d", &num);
ptr = (int*) malloc(num * sizeof(int));//memory allocated using malloc
if(ptr == NULL){
        printf("Error! memory not allocated.");
        exit(0);
}
printf("\nEnter elements of array: ");
for(i = 0; i < num; ++i){
scanf("%d", ptr + i);
```

```
sum += *(ptr + i);
}
printf("\nSum = %d\n", sum);
free(ptr);
return 0;
}
```

Practice 4:

Write a C program to find sum of n elements entered by user while dynamically allocating the memory using calloc() function.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int num, i, *ptr, sum = 0;
printf("Enter number of elements: ");
scanf("%d", &num);
ptr = (int*) calloc(num, sizeof(int));//memory allocated using calloc
if(ptr == NULL){
        printf("Error! memory not allocated.");
        exit(0);
}
printf("\nEnter elements of array: ");
for(i = 0; i < num; ++i){
scanf("%d", ptr + i);
sum += *(ptr + i);
}
printf("\nSum = %d\n", sum);
free(ptr);
return 0;
}
```

Practice 5:

Write a C program to change previously allocated memory using realloc().

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  int *ptr, i , n1, n2;
  printf("Enter size of array: ");
  scanf("%d", &n1);
  ptr = (int*) calloc(n1 , sizeof(int));
  printf("Address of previously allocated memory:\n");
  for(i = 0; i < n1; ++i){
    printf("%x\n",ptr + i);
  }
  printf("\nEnter new size of array:\n");
  scanf("%d", &n2);
   ptr = realloc(ptr, n2);
  for(i = 0; i < n2; ++i){
    printf("%x\n", ptr + i);     return 0;
  }
  return 0;
}
```

- ## Pointer to Pointer:

Syntax:

datatype **identifier;

Practice the following code:

```
#include<stdio.h>
  int main(){
  int i= 3;
  int *ptr;
  int **ptr_ptr;
  ptr = &i;
  ptr_ptr =  &ptr
  printf("Address in Ptr:%x\n", ptr);
  printf("Address of Ptr:%x\n", ptr_ptr);
  printf("Address:%x Content:%d\n", *ptr_ptr,**ptr_ptr);
  return 0;
 }
```

- ## Call by reference:

The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter. To pass a value by reference, argument pointers are passed to the functions just like any other value

Practice 6:

```
#include<stdio.h>
 void change(int *x){
 *x = *x+10;
 }
int main(){
int a = 5;
printf("Before calling change a = %d\n", a);
change(&a);    //Calling function by reference
printf("After calling change a = %d\n", a);
}
```

Do it yourself:

- Write a program in C to add numbers using call by reference
- Write a program in C to store n elements in an array and print the elements using pointer
- Write a program in C to find the largest element using Dynamic Memory Allocation
- Write a program in C to swap elements using call by reference