UNIT-2 (POINTERS) (RAM GOPAL GUPTA- http://ramgopalgupta.com/)

POINTER ARITHMETICS

Basic Concept

To conduct arithmetical operations on pointers is a little different that to conduct them on regular integer data types.

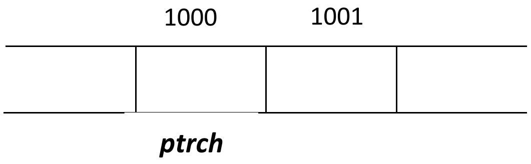
To begin with, only addition and subtraction operations are allowed to be conducted with them, the other makes no sense in the world of pointers. Both addition and subtraction have a different behavior with pointers according the size of data type to which they point.

Suppose we have two pointers:

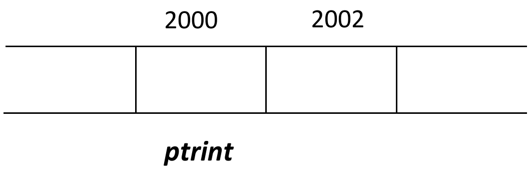
char *ptrch;
int *ptrint;

Assume:

ptrch pointer points to memory location 1000

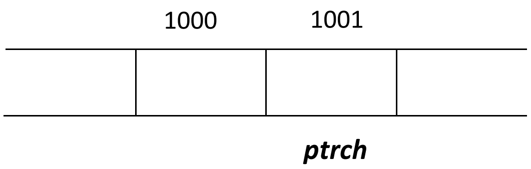


ptrint pointer points to memory location 2000



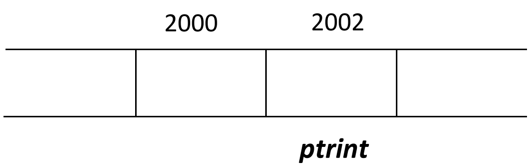
When we write:
ptrch++;   ptrint++;

ptrch pointer points to memory location 1001



Explanation: ptrch++; means ptrch = ptrch + 1;
here ptrch is a char pointer and pointing to address 1000.
The storage capacity of char is 1 byte therefore when the statement ptrch++ executes, it increase the address pointing by the ptrch using given formula:
ptrch = (address contained in ptrch) + 1 x (size of ptrch data type i.e. char)
ptrch = 1000 + 1 x (1)
ptrch = 1001 // new address in ptrch

ptrint pointer points to memory location 2002



Explanation: ptrint++; means ptrint = ptrint + 1;
here ptrch is a int pointer and pointing to address 2000.
The storage capacity of int is 2 byte therefore when the statement ptrint++ executes, it increase the address pointing by the ptrch using given formula:
ptrint = (address contained in ptrint) + 1 x (size of ptrint data type i.e. int)
ptrint = 2000 + 1 x (2)
ptrint = 2002 // new address in ptrint

To make your understanding better in pointer arithmetic here are few more examples:

int i = 12,
int *ip = &i;

char ch = 'a',
*cp = &ch;

Suppose the address of i and ch are 1000, 3000 respectively, therefore ip and cp are at 1000, 3000 initially.

ip holds the memory address 1000 initially

Pointer Arithmetic on int data type	
Pointer Expression	How it is evaluated?
ip = ip + 1	ip = ip + 1 => 1000 + 1*2 => 1002
ip++ or ++ip	ip++ = ip + 1 => 1002 + 1*2 => 1004
ip = ip + 5	ip = ip + 5 => 1004 + 5*2 => 1018
ip = ip – 2	ip = ip - 2 => 1018 - 2*2 => 1014
ip-- or –ip	ip = ip -1 => 1014 - 1*2 => 1012

cp holds the memory address 3000 initially

Pointer Arithmetic on char data type	
Pointer Expression	How it is evaluated?
cp = cp + 1	cp = cp + 1 => 3000 + 1*1 => 3001
cp++ or ++cp	cp = cp + 1 => 3001 + 1*1 => 3002
cp = cp + 5	cp = cp + 5 => 3002 + 5*1 => 3007
cp = cp - 2	cp = cp - 2 => 3007 - 2*1 => 3005
cp-- or –cp	cp = cp -1 => 3005 - 1*1 => 3004

DYNAMIC MEMORY ALLOCATION

As we know, an array is a collection of a fixed number of values. Once the size of an array is declared, we cannot change it.

Sometimes the size of the array we declared may be insufficient. To solve this issue, we can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming.

To allocate memory dynamically, library functions are malloc(), calloc(), realloc() and free() are used. These functions are defined in the <stdlib.h> header file.

malloc() method

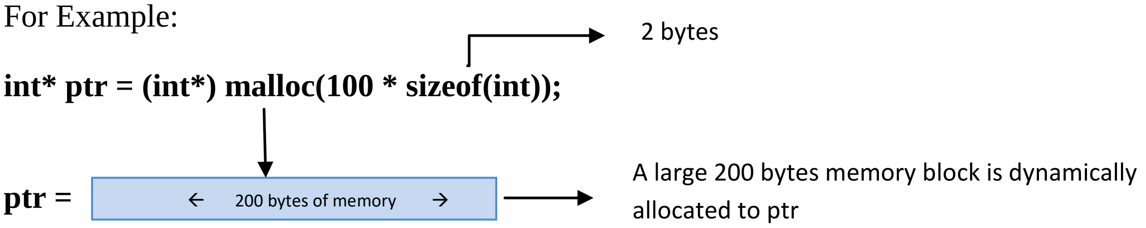
"malloc" or "memory allocation" method in C is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form.

Syntax:

ptr = (cast-type*) malloc(byte-size)

For Example:

int* ptr = (int*) malloc(100 * sizeof(int));


2 bytes
ptr =
200 bytes of memory
A large 200 bytes memory block is dynamically allocated to ptr

sizeof(..) function returns the size of data type which is passed as argument. Since the size of int is 2 bytes, this statement will allocate 200 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.

If space is insufficient, allocation fails and returns a NULL pointer.

Here i am writing a simple program with the help of dynamic memory allocation:
// accept a number and increase it by 5
//try this program and observe the result.

```
main(){

int *ptr = (int*) malloc(sizeof(int));   // Step 1
printf("Enter a Number: ");              // Step 2
scanf("%d",ptr);                         // Step 3
*ptr = *ptr + 5;                         // Step 4
printf("*ptr = %d",*ptr);                // Step 5

}
```

Explanation:

Step 1:- dynamically memory is allocated with memory block of 2 bytes i.e. size of int and the address of such block is assigned to pointer ptr.

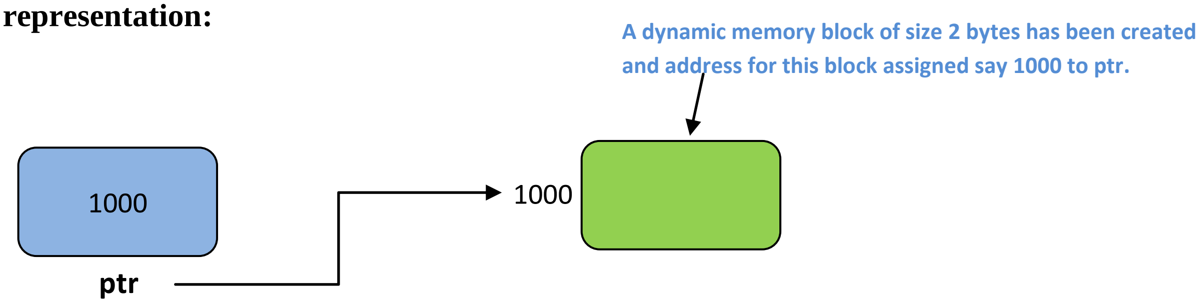
Step 2 & 3:- used to accept a number from the user and store it at the dynamic memory address which is hold by pointer ptr.

Step 4:- *ptr = *ptr +5; right side of this expression first fetch the value from memory address stored in pointer ptr then add the value 5 in that and again store the new value in the same address held by ptr.

Step 5:- display the value from address stored in ptr.

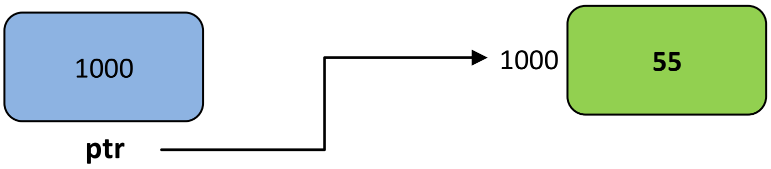
Pictorial representation:

Step 1:-

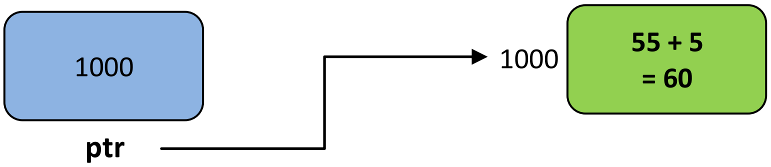

A dynamic memory block of size 2 bytes has been created and address for this block assigned say 1000 to ptr.

Step 2 & 3:-

Enter a number: 55



Step 4:-



Step 5:-

Display the value stored at memory address 1000

*ptr = 60

I am giving two programs here; both will give the same result. You only observer the coding style and on this basis try some simple program from your own using DMA.

```
#include<stdio.h>

void main(){

        int num;

        int *ptr;

        ptr=&num;

        printf("Enter a Number:");

        scanf("%d",ptr);

        printf("Square of Number is %d\n",(*ptr * *ptr));

}
```

```
#include<stdio.h>

#include<stdlib.h>

void main(){

        int *ptr;

        ptr=(int*) malloc(sizeof(int));

        printf("Enter a Number:");

        scanf("%d",ptr);

        printf("Square of Number is %d\n",(*ptr * *ptr));

}
```

calloc() method

“calloc” or “contiguous allocation” method in C is used to dynamically allocate the specified number of blocks of memory of the specified type. It initializes each block with a default value ‘0’.

Syntax:

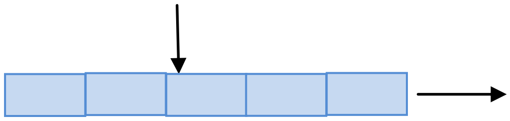
ptr = (cast-type*) calloc(n, element-size);

For Example:

2 bytes

int* ptr = (int*) calloc(5 * sizeof(int));

ptr =



5 blocks of 2 bytes each is dynamically allocated to ptr

sizeof(..) function returns the size of data type which is passed as argument. Since the size of int is 2 bytes, this statement allocates contiguous space in memory for 5 elements each with the size of the int.

If space is insufficient, allocation fails and returns a NULL pointer.

// accept a number and increase it by 5
//try this program and observe the result.

```
main(){

int *ptr = (int*) calloc(1 * sizeof(int));          // Step 1
printf(“Enter a Number: ”);                // Step 2
scanf(“%d”,ptr);                           // Step 3
*ptr = *ptr + 5;                           // Step 4
printf(“*ptr = %d”,*ptr);                  // Step 5

}
```

free() method

“free” method in C is used to dynamically de-allocate the memory. The memory allocated using functions malloc() and calloc() is not de-allocated on their own. Hence the free() method is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

Syntax:

free(ptr);

realloc() method

“realloc” or “re-allocation” method in C is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to dynamically re-allocate memory.

Syntax:

ptr = realloc(ptr, newSize);

where ptr is reallocated with new size 'newSize'.

ARRAY AND POINTERS:

Dynamic size array implementation with malloc DMA (dynamic memory allocation)

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        // This pointer will hold the
        // base address of the block created
        int* ptr;
        int n, i;
        // Get the number of elements for the array
        n = 5;
        printf("Enter number of elements: %d\n", n);

        // Dynamically allocate memory using malloc()
        ptr = (int*)malloc(n * sizeof(int));

        // Check if the memory has been successfully
        // allocated by malloc or not
        if (ptr == NULL) {
                printf("Memory not allocated.\n");
                exit(0);
        }
        else {

                // Memory has been successfully allocated
                printf("Memory successfully allocated using malloc.\n");
                // Get the elements of the array
                for (i = 0; i < n; ++i) {
                        ptr[i] = i + 1;
                }
                // Print the elements of the array
                printf("The elements of the array are: ");
                for (i = 0; i < n; ++i) {
                        printf("%d, ", ptr[i]);
                }
        }
        return 0;
}
```

Output:

```
Enter number of elements: 5

Memory successfully allocated using malloc.

The elements of the array are: 1, 2, 3, 4, 5,
```

Dynamic size array implementation with calloc DMA (dynamic memory allocation)

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        // This pointer will hold the
        // base address of the block created
        int* ptr;
        int n, i;

        // Get the number of elements for the array
        n = 5;
        printf("Enter number of elements: %d\n", n);

        // Dynamically allocate memory using calloc()
        ptr = (int*)calloc(n, sizeof(int));

        // Check if the memory has been successfully
        // allocated by calloc or not
        if (ptr == NULL) {
                printf("Memory not allocated.\n");
                exit(0);
        }
        else {
                // Memory has been successfully allocated
                printf("Memory successfully allocated using calloc.\n");

                // Get the elements of the array
                for (i = 0; i < n; ++i) {
                        ptr[i] = i + 1;
                }
                // Print the elements of the array
                printf("The elements of the array are: ");
                for (i = 0; i < n; ++i) {
                        printf("%d, ", ptr[i]);
                }
        }

        return 0;
}
```

Output:

```
Enter number of elements: 5

Memory successfully allocated using calloc.

The elements of the array are: 1, 2, 3, 4, 5,
```

EXAMPLE PROGRAM CODE OF FREE() FUNCTION

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	// This pointer will hold the
	// base address of the block created
	int *ptr, *ptr1;
	int n, i;

	// Get the number of elements for the array
	n = 5;
	printf("Enter number of elements: %d\n", n);
	// Dynamically allocate memory using malloc()
	ptr = (int*)malloc(n * sizeof(int));
	// Dynamically allocate memory using calloc()
	ptr1 = (int*)calloc(n, sizeof(int));
	// Check if the memory has been successfully
	// allocated by malloc or not
	if (ptr == NULL || ptr1 == NULL) {
		printf("Memory not allocated.\n");
		exit(0);
	}
	else {
		// Memory has been successfully allocated
		printf("Memory successfully allocated using malloc.\n");
		// Free the memory
		free(ptr);
		printf("Malloc Memory successfully freed.\n");
		// Memory has been successfully allocated
		printf("\nMemory successfully allocated using calloc.\n");
		// Free the memory
		free(ptr1);
		printf("Calloc Memory successfully freed.\n");
	}
	return 0;
}
```

Output:

```
Enter number of elements: 5
Memory successfully allocated using malloc.
Malloc Memory successfully freed.

Memory successfully allocated using calloc.
Calloc Memory successfully freed.
```

EXAMPLE PROGRAM CODE OF REALLOC() FUNCTION

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	// This pointer will hold the
	// base address of the block created
	int* ptr;
	int n, i;
	// Get the number of elements for the array
	n = 5;
	printf("Enter number of elements: %d\n", n);
	// Dynamically allocate memory using calloc()
	ptr = (int*)calloc(n, sizeof(int));
	// Check if the memory has been successfully
	// allocated by malloc or not
	if (ptr == NULL) {
		printf("Memory not allocated.\n");
		exit(0);
	}
	else {
		// Memory has been successfully allocated
		printf("Memory successfully allocated using calloc.\n");
		// Get the elements of the array
		for (i = 0; i < n; ++i) {
			ptr[i] = i + 1;
		}
		// Print the elements of the array
		printf("The elements of the array are: ");
		for (i = 0; i < n; ++i) {
			printf("%d, ", ptr[i]);
		}
		// Get the new size for the array
		n = 10;
		printf("\n\nEnter the new size of the array: %d\n", n);
		// Dynamically re-allocate memory using realloc()
		ptr = realloc(ptr, n * sizeof(int));
		// Memory has been successfully allocated
		printf("Memory successfully re-allocated using realloc.\n");

		// Get the new elements of the array
		for (i = 5; i < n; ++i) {
			ptr[i] = i + 1;
		}
```

```
        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }
        free(ptr);
    }
    return 0;
}
```

Output:

```
Enter number of elements: 5
Memory successfully allocated using calloc.
The elements of the array are: 1, 2, 3, 4, 5,


Enter the new size of the array: 10
Memory successfully re-allocated using realloc.
The elements of the array are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

For more details visit: https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/