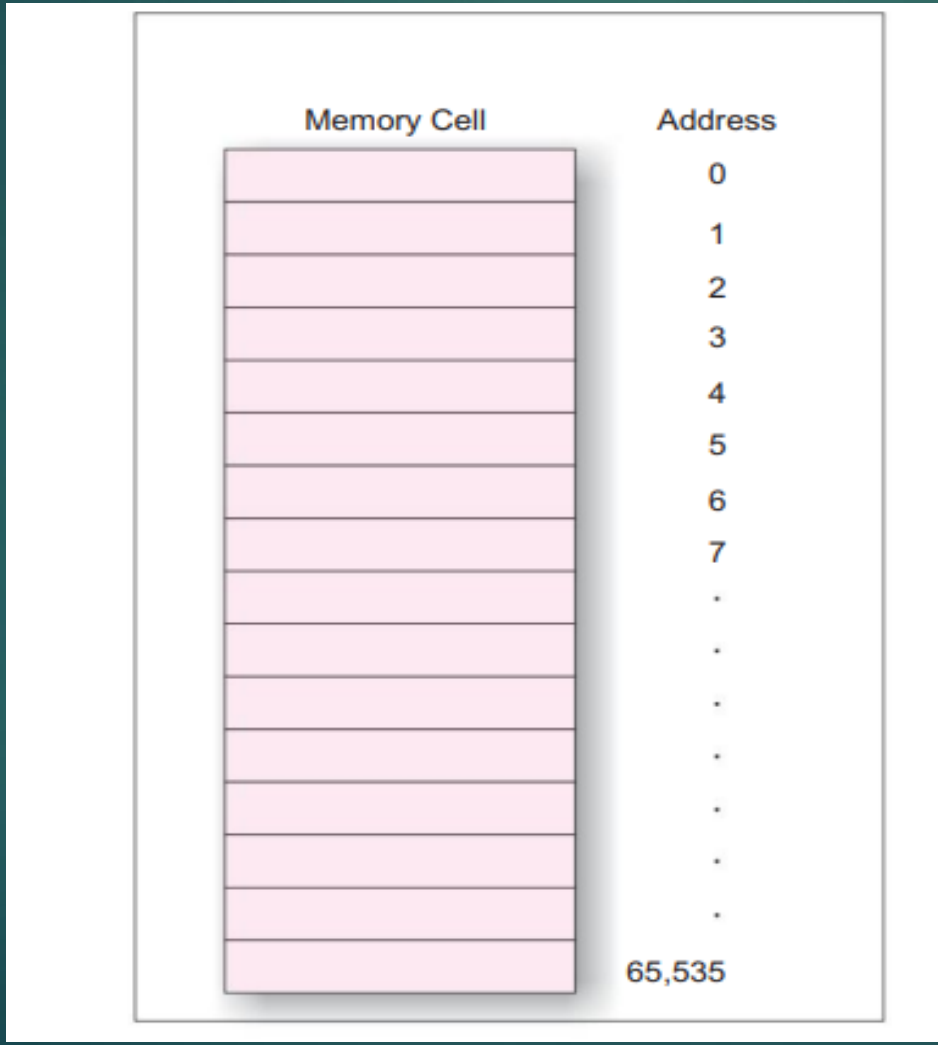
POINTERS

MODULE 5

Memory organization

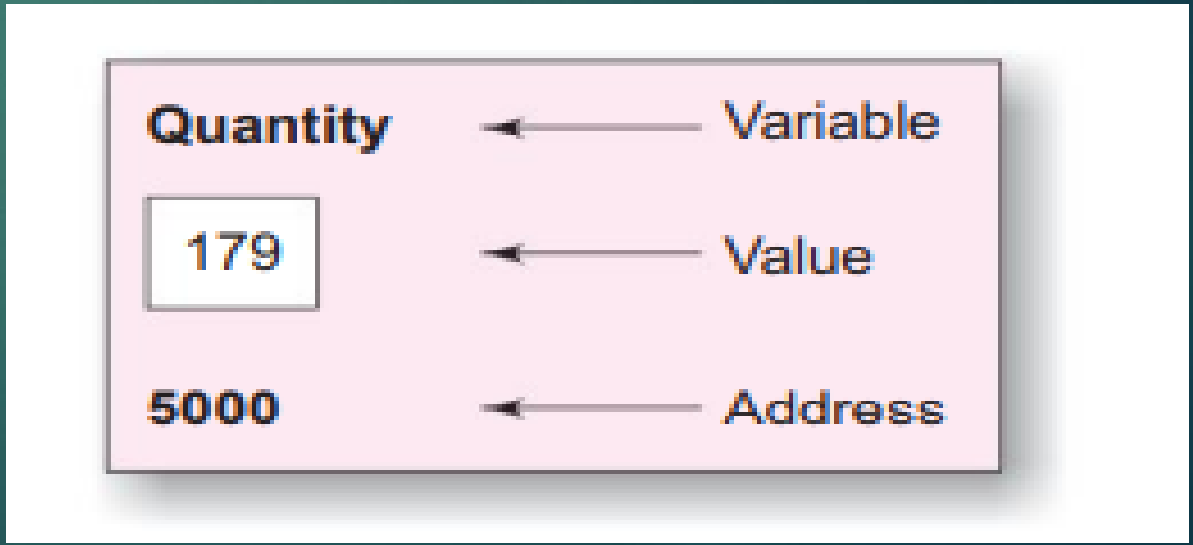


Pointer

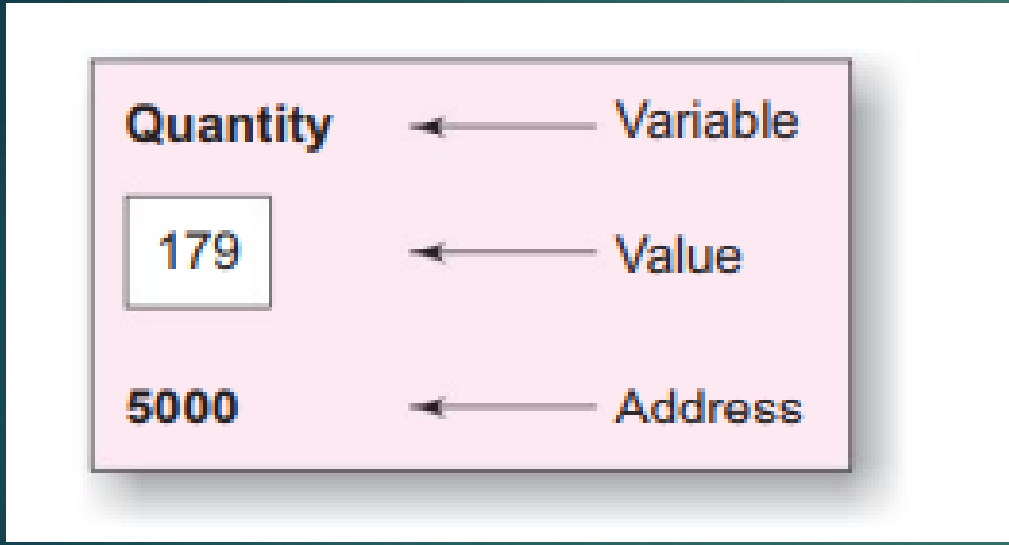
- Variable that holds the memory address of the location of another variable in the memory.
- It is a derived data type.

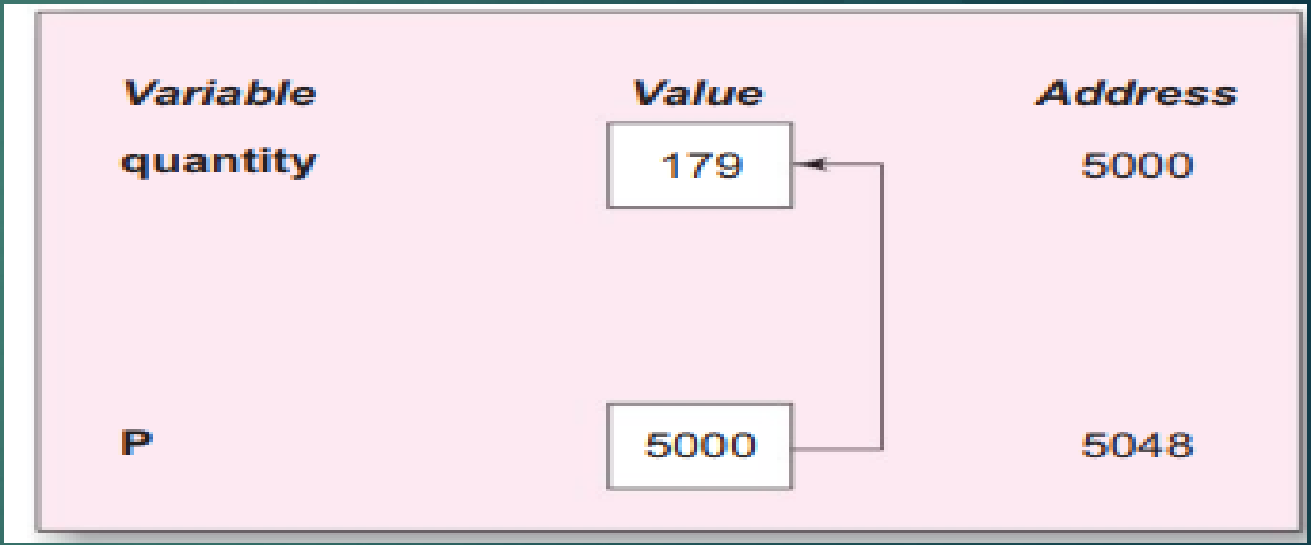
Example:

int Quantity = 179;



Pointer Variable





Declaration

syntax: *datatype * pointer name;*

Pointer variable

Holds the address of another variable that is of the specified datatype given

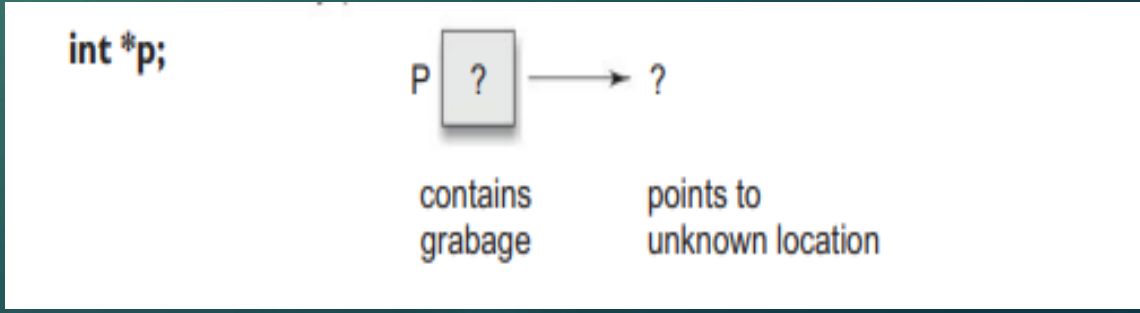
Pointer variable

datatype * pointer name

- * tells that the variable pointer name is a pointer variable
- Pointer name needs a memory location
- Pointer name points to variable of type data type.

Example : int *p; // integer pointer

float *p // float pointer



Declaration styles

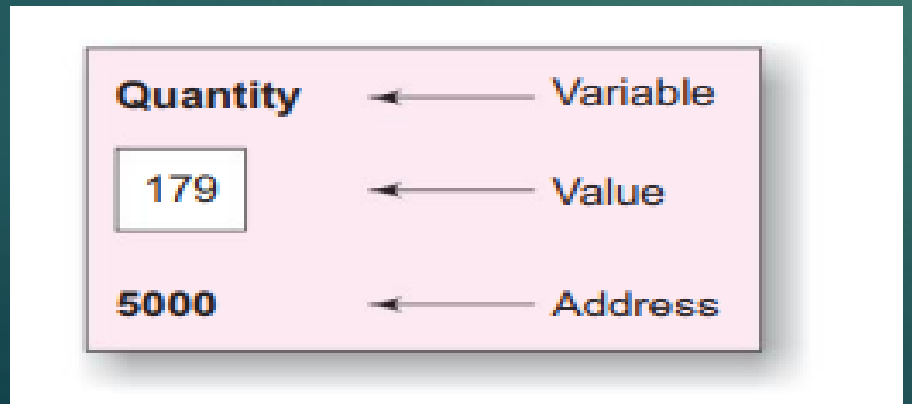
- Declared similarly as normal variables except for the addition of the unary operator(*).
- * can appear anywhere between type name and the pointer variable name.

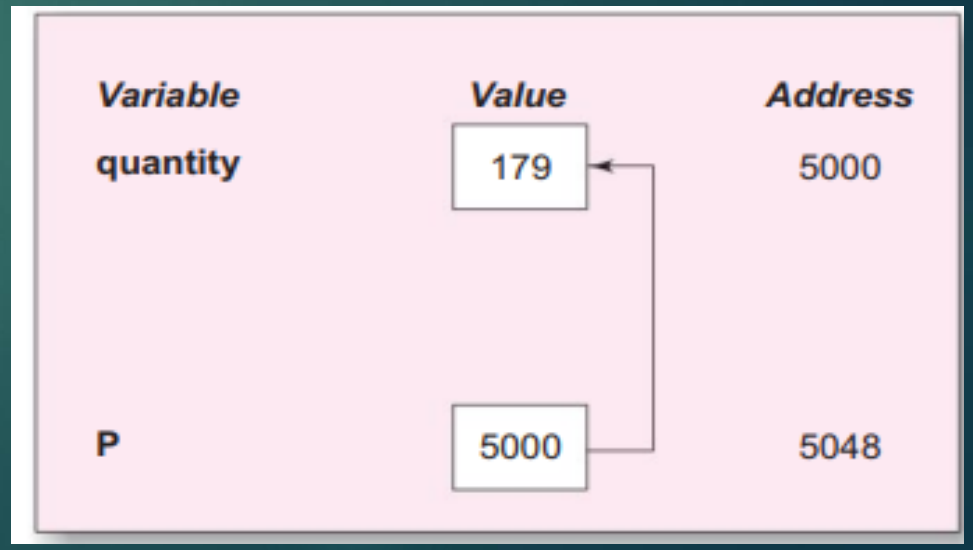
```
int*      p;    //Style 1
int* p;   //Style 2
int       * p;  // Style 3
```

Access the address of a variable

- Using & operator available in C.
- The operator & immediately preceding a variable returns the address of the variable associated with it.

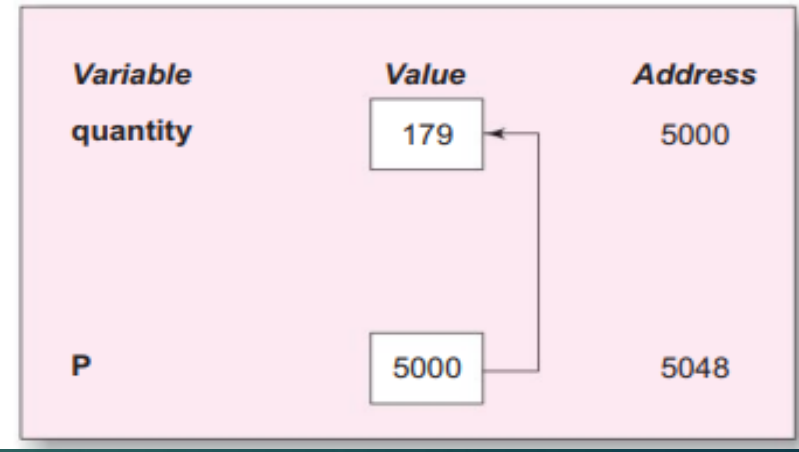
Example : p = &quantity;





Initialize a pointer

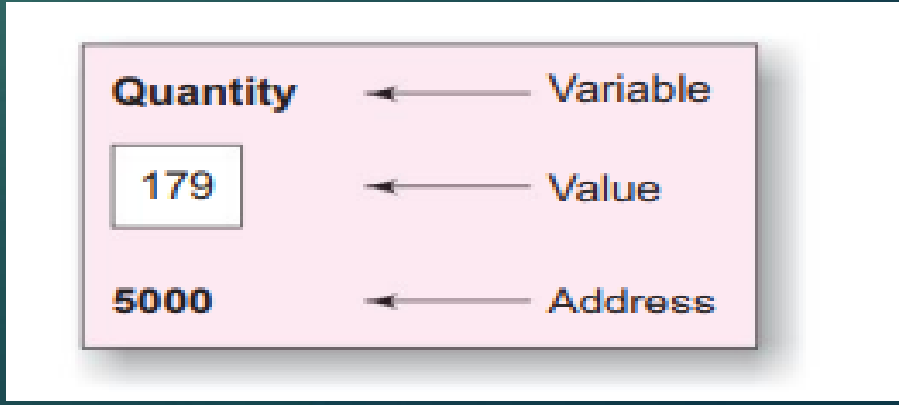
- Assigning address of a variable to a pointer variable.
- Use assignment operator for initialization.



Example: int quantity;

int *p; /*declaration*/

p = &quantity /*initialization*/



- Combination of declaration of data variable, declaration of pointer variable and initialization of pointer variable in single step.

Pointer variable

Example: *int x , *p = &x;*

Initialize p to address of x

Declaration

- Declare pointer variable with an initial value of NULL or zero is also possible.

```
int *p = NULL;
int *p = 0;
```

Accessing a variable through its pointer

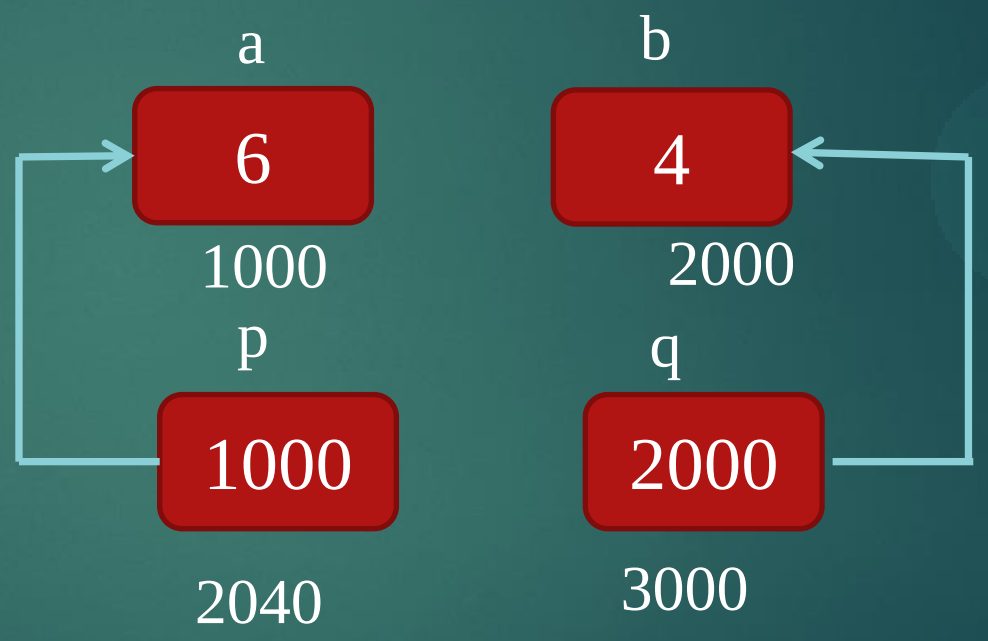
- Using unary operator * called indirection or dereferencing.

Example :

```
int quantity, *p, n;
quantity = 179;
p = &quantity; // address of variable quantity
n = *p; //Returns value at address
```

Example

```
int a= 6, b=4;
int *p,*q;
p=&a;
q=&b;
printf("Value of a = %d",*p);
```

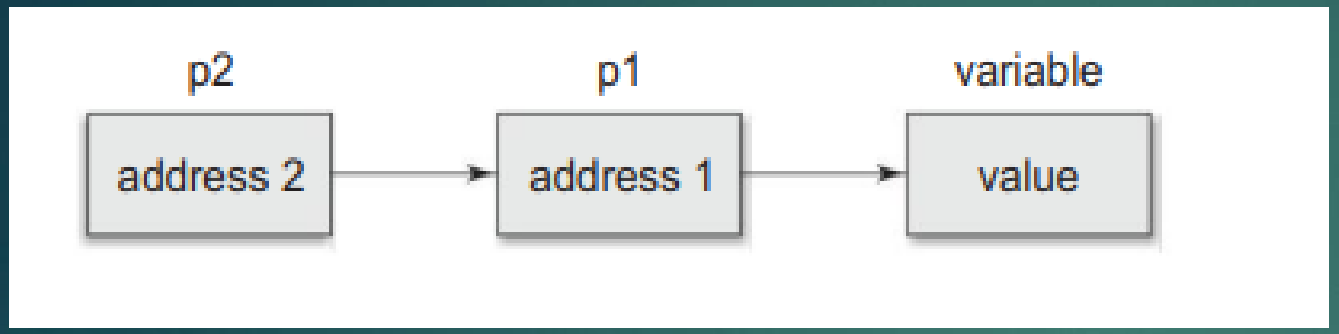


Example: Predict the output

```
int a= 6, b=4;
int *p,*q;
p=&a;
q=&b;
printf("value of a = %d",*p);
printf("address of a = %d",&a);
printf("address of a = %d",p);
printf("address of p = %d",&p);
```

```
void main()
 { int x, y;
 int *ptr;
 x = 10;
 ptr = &x;
 y = *ptr;
 printf("Value of x is %d\n\n",x);
 printf("%d is stored at addr %u\n", x, &x);
 printf("%d is stored at addr %u\n", *&x, &x);
 printf("%d is stored at addr %u\n", *ptr, ptr);
 printf("%d is stored at addr %u\n", ptr, &ptr);
 printf("%d is stored at addr %u\n", y, &y);
 *ptr = 25;
 printf("\nNow x = %d\n",x);
}
```

Chain of Pointers



- A variable that is a pointer to a pointer must be declared as:

*int **p2*

```
main ( )
 {
 int x, *p1, **p2;
 x = 100;
 p1 = &x;
p2 = &p1
printf ("%d", **p2);
 }
```

Pointer Expressions

- C allows us to add integers to or subtract integers from pointers, as well as to subtract one pointer from another.

- Pointers can also be compared using the relational operators.

Sample Programs

- Program to compute the sum of two numbers using the concept of pointers
- Program to compute the largest of three numbers

Elements of array stored

```c

#include<stdio.h>

int main()
{
    int arr[5] = {1, 2, 3, 4, 5}, i;

    for(i = 0; i < 5; i++)
    {
        printf("Value of arr[%d] = %d\t", i, arr[i]);
        printf("Address of arr[%d] = %u\n", i, &arr[i]);
    }

    return 0;
}
```

```
Value of arr[0] = 1     Address of arr[0] = 1297118080
Value of arr[1] = 2     Address of arr[1] = 1297118084
Value of arr[2] = 3     Address of arr[2] = 1297118088
Value of arr[3] = 4     Address of arr[3] = 1297118092
Value of arr[4] = 5     Address of arr[4] = 1297118096
```

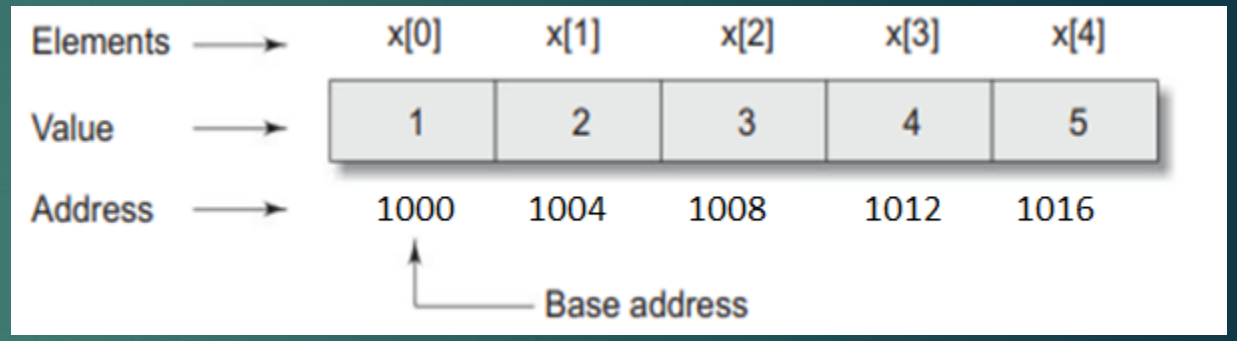
Using pointers to access elements and address of elements in an array

```c

#include<stdio.h>

int main()
{
    int arr[5] = {1, 2, 3, 4, 5}, i;

    for(i = 0; i < 5; i++)
    {
        printf("Value of a[%d] = %d\t", i, *(arr + i) );
        printf("Address of a[%d] = %u\n", i, arr + i );
    }

    return 0;
}
```

```
Value of a[0] = 1       Address of a[0] = 4063571264
Value of a[1] = 2       Address of a[1] = 4063571268
Value of a[2] = 3       Address of a[2] = 4063571272
Value of a[3] = 4       Address of a[3] = 4063571276
Value of a[4] = 5       Address of a[4] = 4063571280
```

Pointers and Arrays

```
int x[5] = {1, 2, 3, 4, 5};
int *p;
p = x; / p = &x[0];
```



Assigning 1-D array to a Pointer variable

```c
#include<stdio.h>

int main()
{
    int arr[5] = {1, 2, 3, 4, 5}, i;
    int *p;
    p = arr;
    for(i = 0; i < 5; i++)
    {
        printf("Value of a[%d] = %d\t", i, *(p + i) );
        printf("Address of a[%d] = %u\n", i, p + i );
    }
    return 0;
}
```

```
Value of a[0] = 1       Address of a[0] = 2432394160
Value of a[1] = 2       Address of a[1] = 2432394164
Value of a[2] = 3       Address of a[2] = 2432394168
Value of a[3] = 4       Address of a[3] = 2432394172
Value of a[4] = 5       Address of a[4] = 2432394176
```

Pointer Expressions

- C allows us to add integers to or subtract integers from pointers, as well as to subtract one pointer from another.

Example: p1 + 4

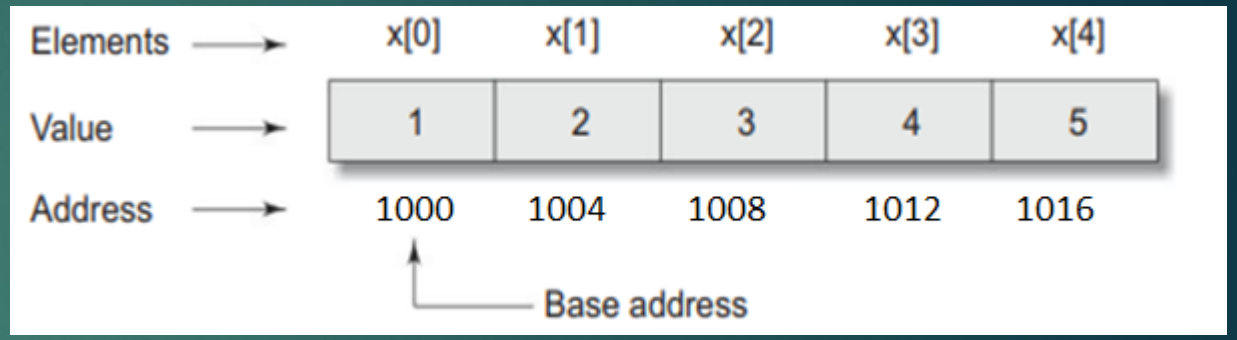
p2 – 2

p1 – p2

Increment/Decrement

- **Post Increment**

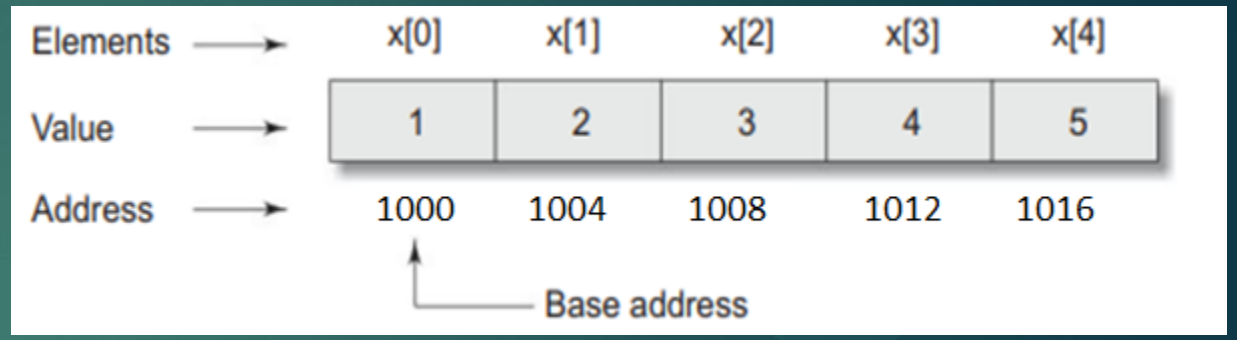
```
int x[5] = {1, 2, 3, 4, 5};
 int *p;
 p = x;
 p++ ;
printf("%d",*p++);
printf("%d",*p);
```



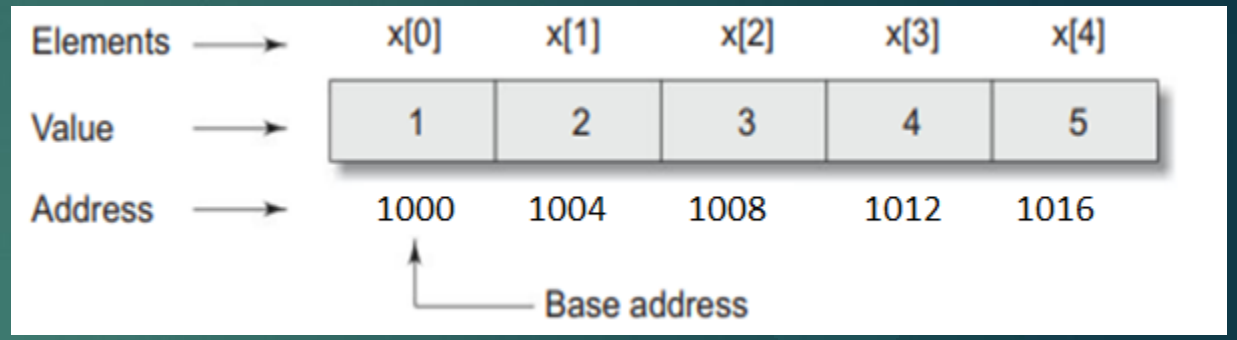
Increment/Decrement

- **Pre Increment**

```
int x[5] = {1, 2, 3, 4, 5};
 int *p;
 p = x;
 ++p ;
printf("%d",*++p);
printf("%d",*p);
```



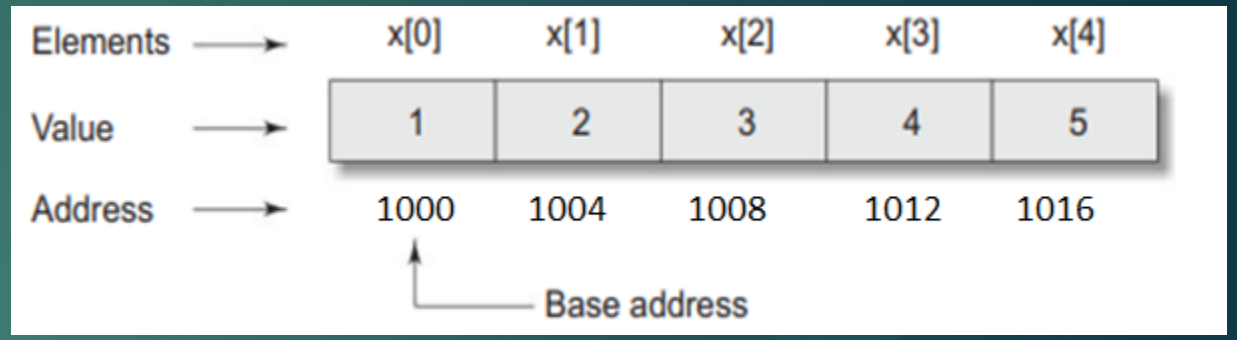
```
int x[5] = {1, 2, 3, 4, 5};
 int *p;
 p = x;
 p + 2 ;
 p + 3;
printf("%d",*(p+2));
 printf("%d",*(p+3));
```



- Write a program using pointers to compute the sum of all elements stored in an array.

```
main()
{
int *p, sum, i;
int x[5] = {5,9,6,3,7};
i = 0;
p = x; /* initializing with base address of x */
printf("Element Value Address\n\n");
while(i < 5)   or // for (i=x; i<=x+5;i++)
{
printf(" x[%d] %d %u\n", i, *p, p);
sum = sum + *p; /* accessing array element */
i++, p++; /* incrementing pointer */
}
printf("\n Sum = %d\n", sum);
printf("\n &x[0] = %u\n", &x[0]);
printf("\n p = %u\n", p);
}
```

Using pointers to access elements and address of array



- *(p+0)
- *(p+1)
- *(p+2)
- *(p+3)
- *(p+4) → *(p+i)
- One dimensional array

(x+i) /(p+i)

- We can use a array names as pointers but assigning a new address to them is not possible.
- Example:

```
int main()
{
int a[]={1,2,3,4,5};
printf("%p",a++);
return 0;
}
```

a=a+1

Pointer to an Array *int(*p)[5]*

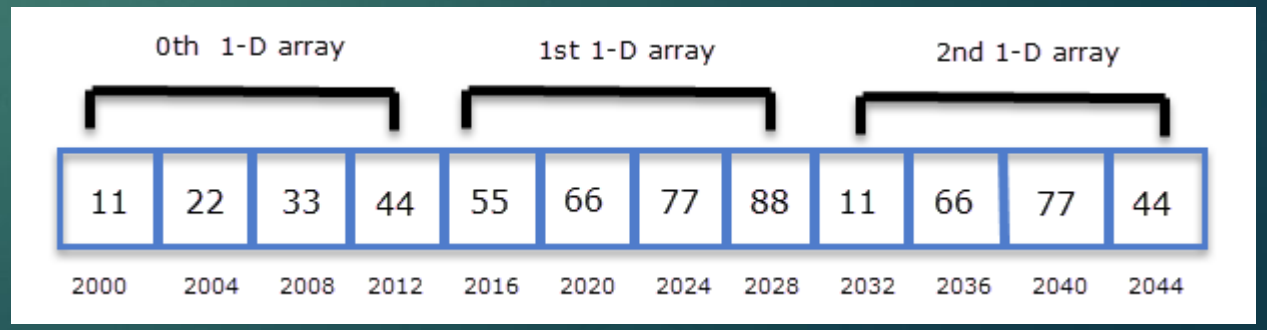
```c
#include<stdio.h>

int main()
{
    int *p; // pointer to int
    int (*parr)[5]; // pointer to an array of 5 integers
    int arr[5]; // an array of 5 integers

    p = arr;
    parr = arr;

    printf("Address of p = %u\n", p );
    printf("Address of parr = %u\n", parr );

    p++;
    parr++;

    printf("\nAfter incrementing p and parr by 1 \n\n");
    printf("Address of p = %u\n", p );
    printf("Address of parr = %u\n", parr );
    printf("Address of parr = %u\n", *parr );
return 0;
}
```

```
Address of p = 1282539200
Address of parr = 1282539200

After incrementing p and parr by 1

Address of p = 1282539204
Address of parr = 1282539220
Address of parr = 1282539220
```

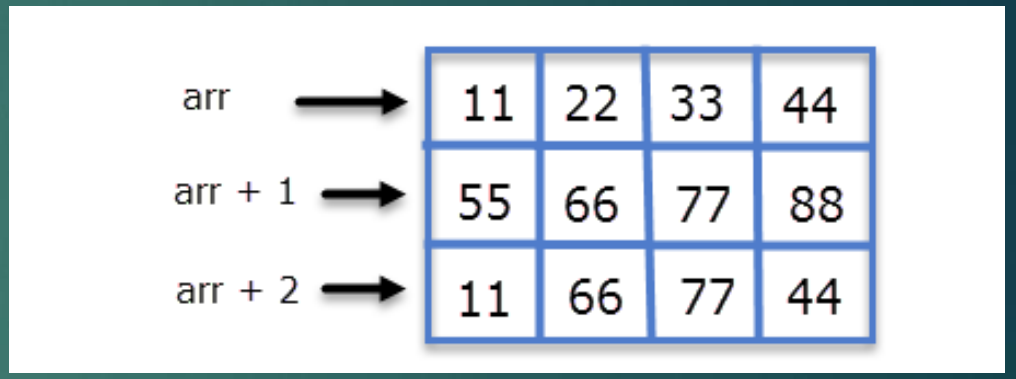
Pointers and 2D Arrays

```
int arr[3][4] = { {11,22,33,44},
                  {55,66,77,88},
                  {11,66,77,44} };
```

	Col 0	Col 1	Col 2	Col 3
Row 0	11	22	33	44
Row 1	55	66	77	88
Row 2	11	66	77	44



- arr points to 0th 1-D array.
  (arr + 1) points to 1st 1-D array.
  (arr + 2) points to 2nd 1-D array.



(arr + i) points to ith 1-D array
*(arr+i) points to the base address of the ith 1-D array.

- *(arr + i) points to the address of the 0th element of the 1-D array.
  *(arr + i) + 1 points to the address of the 1st element of the 1-D array
  *(arr + i) + 2 points to the address of the 2nd element of the 1-D array
- Hence we can conclude that:
- *(arr + i) + j points to the base address of jth element of ith 1-D array.
- On dereferencing *(arr + i) + j we will get the value of jth element of ith 1-D array.

(*(*(arr + i) + j))

```c
#include<stdio.h>

int main()
{
    int arr[3][4] = {
                        {11,22,33,44},
                        {55,66,77,88},
                        {11,66,77,44}
                    };

    int i, j;

    for(i = 0; i < 3; i++)
    {
        printf("Address of %d th array %u \n",i , *(arr + i));
        for(j = 0; j < 4; j++)
        {
            printf("arr[%d][%d]=%d\n", i, j, *( *(arr + i) + j) );
        }
        printf("\n\n");
    }
    return 0;
}
```

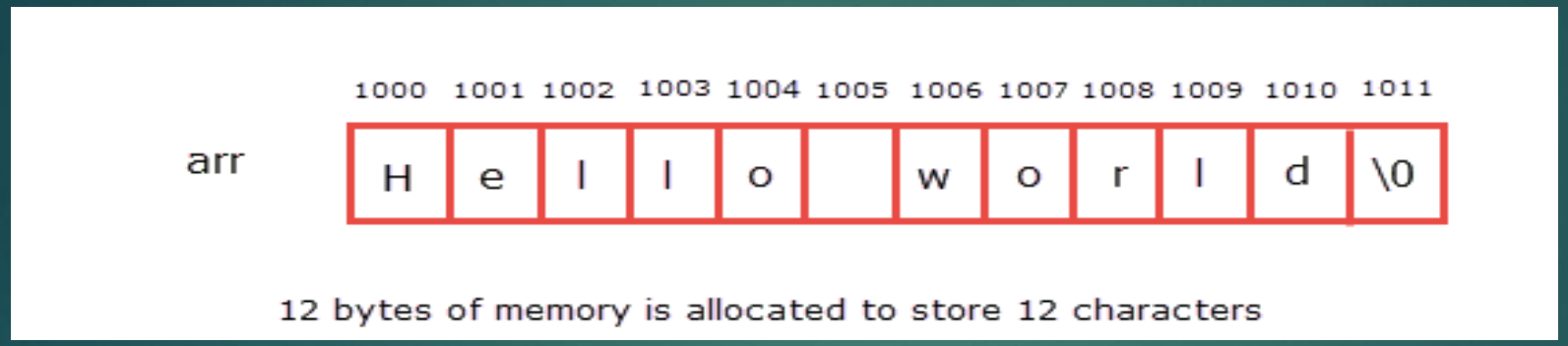
```
Address of 0 th array 3608971664
arr[0][0]=11
arr[0][1]=22
arr[0][2]=33
arr[0][3]=44


Address of 1 th array 3608971680
arr[1][0]=55
arr[1][1]=66
arr[1][2]=77
arr[1][3]=88


Address of 2 th array 3608971696
arr[2][0]=11
arr[2][1]=66
arr[2][2]=77
arr[2][3]=44
```

```c
#include<stdio.h>

int main()
{
    int arr[3][4] = {
                        {11,22,33,44},
                        {55,66,77,88},
                        {11,66,77,44}
                    };
    int i, j;
    int (*p)[4];
    p = arr;
    for(i = 0; i < 3; i++)
    {
        printf("Address of %d th array %u \n",i , *(p + i));
        for(j = 0; j < 4; j++)
        {
            printf("arr[%d][%d]=%d\n", i, j, *( *(p + i) + j) );
        }
        printf("\n\n");
    }
    return 0;
}
```

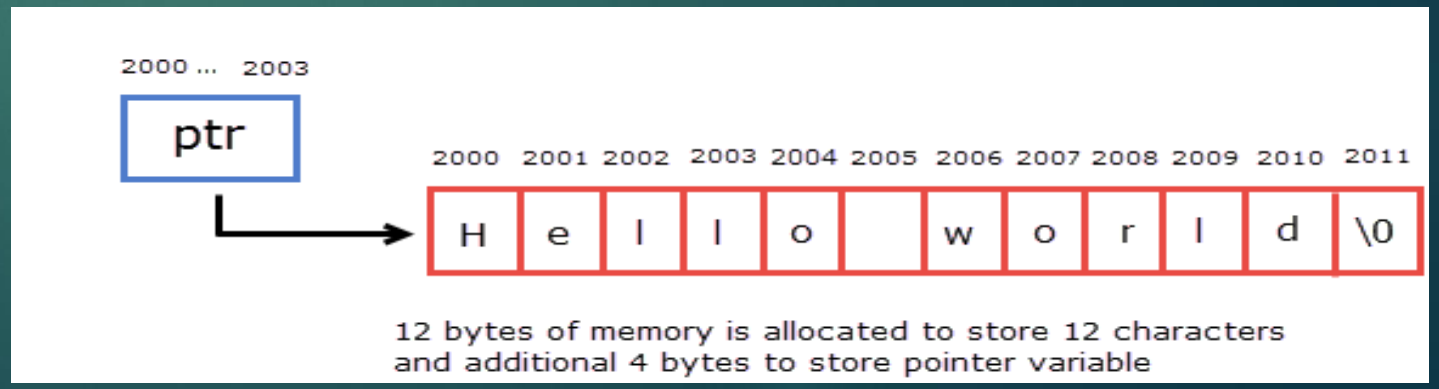
```
Address of 0 th array 2115621456
arr[0][0]=11
arr[0][1]=22
arr[0][2]=33
arr[0][3]=44


Address of 1 th array 2115621472
arr[1][0]=55
arr[1][1]=66
arr[1][2]=77
arr[1][3]=88


Address of 2 th array 2115621488
arr[2][0]=11
arr[2][1]=66
arr[2][2]=77
arr[2][3]=44
```

Pointer and character arrays

char arr[] = "Hello World"; // array version

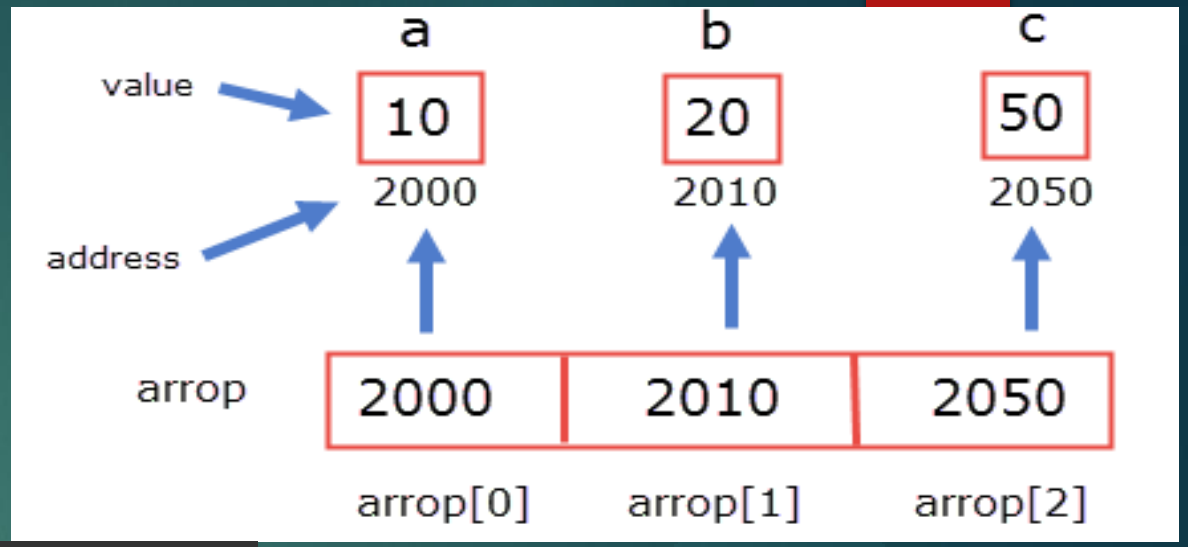


char ptr* = "Hello World"; // pointer version



Array of pointers

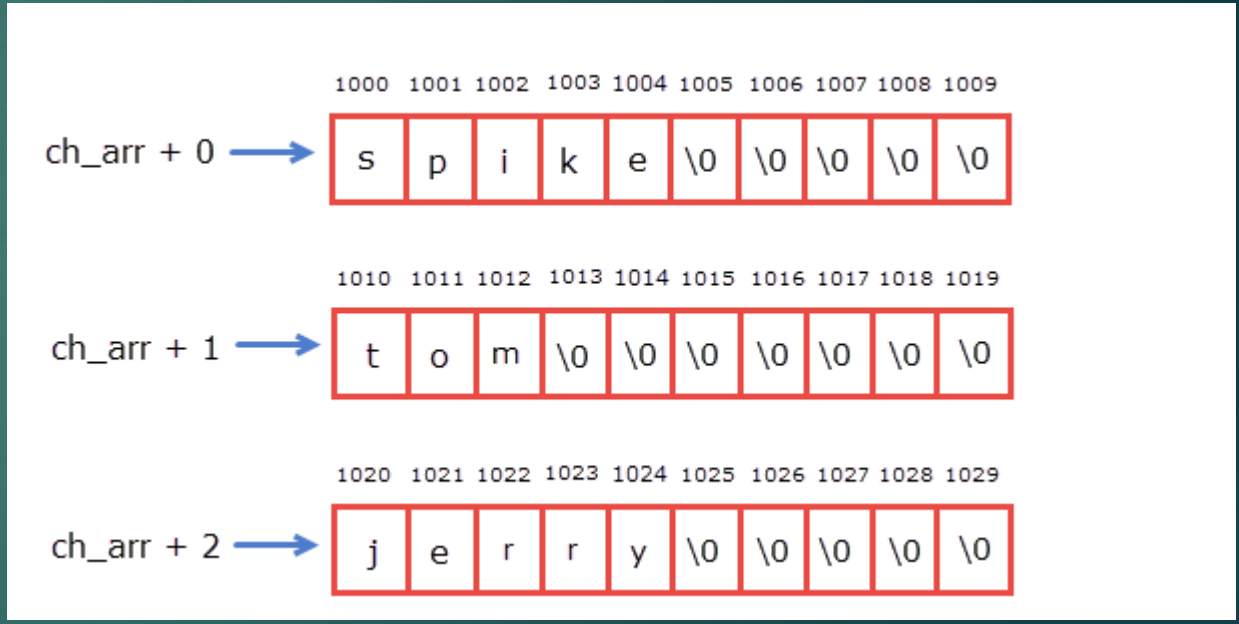
Syntax : datatype *array_name[size];




```c
#include<stdio.h>

int main()
{
    int *arrop[3];
    int a = 10, b = 20, c = 50, i;

    arrop[0] = &a;
    arrop[1] = &b;
    arrop[2] = &c;

    for(i = 0; i < 3; i++)
    {
        printf("Address = %d\t Value = %d\n", arrop[i], *arrop[i]);
    }

    return 0;
}
```

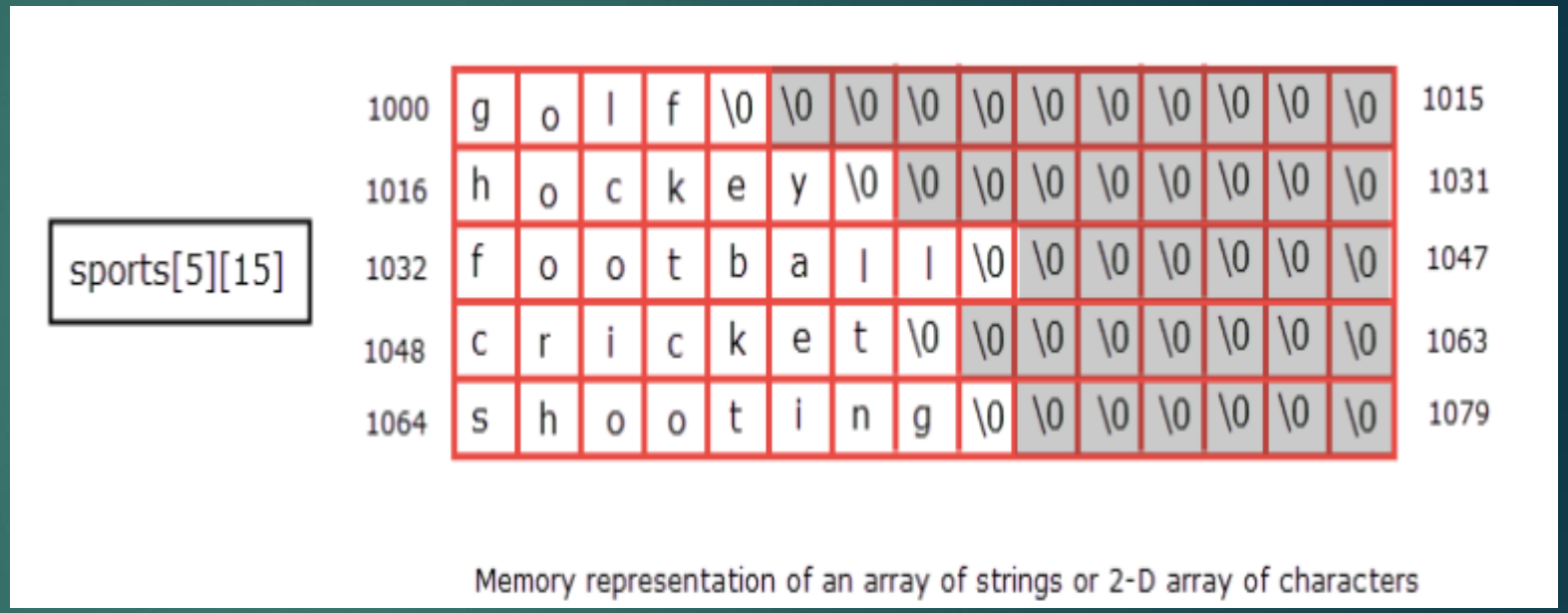
Array of Strings

```
char arr [3][10] = {
                    "spike",
                     "tom",
                    "jerry"
                     };
```



Array of Pointers to Strings

```
char sports[5][15] = {
                "golf",
                "hockey",
                "football",
                "cricket",
                "shooting"
            };
```



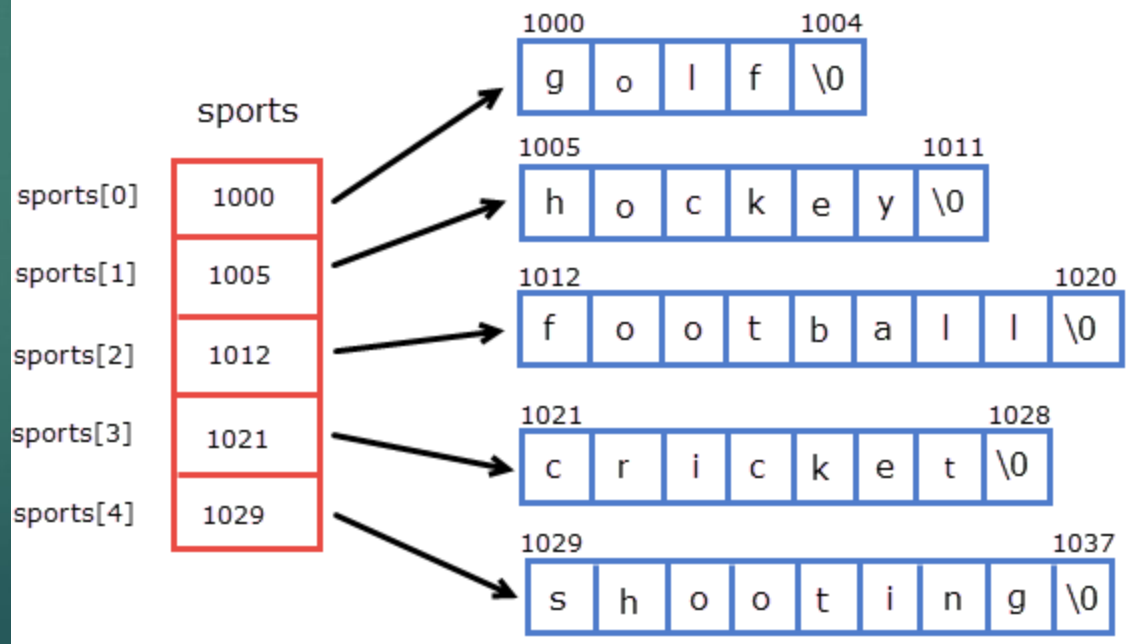
Memory representation of an array of strings or 2-D array of characters

Array of pointers:

- An array of character pointers where each pointer points to the first character of the string or the base address of the string.

- Declaration and Initialization:

```
char *sports[5] = {
                "golf",
                "hockey",
                "football",
                "cricket",
                "shooting"
            };
```



Memory representation of array of pointers

```c
#include <stdio.h>

int main(void) {
  char *sports[5] = {
                    "golf",
                    "hockey",
                    "football",
                    "cricket",
                    "shooting"
                  };
  int r, c;
  for (r = 0; r < 5; r++) {
    c = 0;
    while(*(sports[r] + c) != '\0') {
      printf("sports[%d] is stored at %d\n and value is %c\n", r,(sports[r] + c),*(sports[r] + c));
      c++;
    }
    printf("\n");
  }

  return 0;
}
```

```
sports[0] is stored at 4196056
 and value is g
sports[0] is stored at 4196057
 and value is o
sports[0] is stored at 4196058
 and value is l
sports[0] is stored at 4196059
 and value is f
```

```
sports[1] is stored at 4196061
 and value is h
sports[1] is stored at 4196062
 and value is o
sports[1] is stored at 4196063
 and value is c
sports[1] is stored at 4196064
 and value is k
sports[1] is stored at 4196065
 and value is e
sports[1] is stored at 4196066
 and value is y
```

```
sports[2] is stored at 4196068
 and value is f
sports[2] is stored at 4196069
 and value is o
sports[2] is stored at 4196070
 and value is o
sports[2] is stored at 4196071
 and value is t
sports[2] is stored at 4196072
 and value is b
sports[2] is stored at 4196073
 and value is a
sports[2] is stored at 4196074
 and value is l
sports[2] is stored at 4196075
 and value is l
```

```
sports[3] is stored at 4196077
 and value is c
sports[3] is stored at 4196078
 and value is r
sports[3] is stored at 4196079
 and value is i
sports[3] is stored at 4196080
 and value is c
sports[3] is stored at 4196081
 and value is k
sports[3] is stored at 4196082
 and value is e
sports[3] is stored at 4196083
 and value is t
```

```
sports[4] is stored at 4196085
 and value is s
sports[4] is stored at 4196086
 and value is h
sports[4] is stored at 4196087
 and value is o
sports[4] is stored at 4196088
 and value is o
sports[4] is stored at 4196089
 and value is t
sports[4] is stored at 4196090
 and value is i
sports[4] is stored at 4196091
 and value is n
sports[4] is stored at 4196092
 and value is g
```