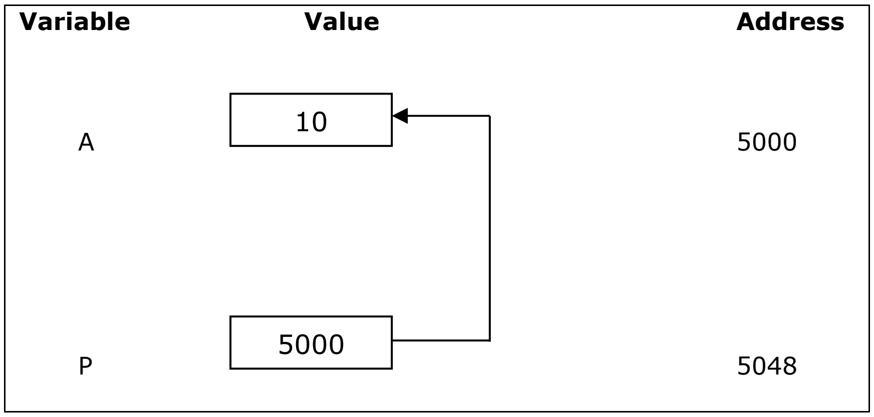
1 What is pointer? How to declare and initialize it?

- A pointer is a variable that contains address or location of another variable.
- Pointer is a derived data type in C.
- Pointers contain memory address as their values, so they can also be used to access and manipulate data stored in memory.

  Example:
  ```
  void main()
  {
        int a=10, *p;
        p = &a;
        printf(“%d %d %d”, a, *p, p);
  }
  ```
  Output: 10 10 5000

- p is integer pointer variable.
- & is address of or referencing operator which returns memory address of variable.
- is indirection or dereferencing operator which returns value stored at that memory address.
- & operator is the inverse of * operator (x = a is same as x = *(&a)).



Declaration of pointer:

Syntax: data_type *pt_name ;

Example: int *p, float *q, char *c ;

1) The asterisk (*) tells that the variable **pt_name** is a pointer variable.
2) pt_name needs a memory location to store address of another variable.
3) pt_name points to a variable of type **data_type.**

Initialization of the pointer:

```
int  a=5, x, *p; // Declares pointer variable p and regular variable a and x
p = &a           // Initializes p with address of a
x = *p;          // p contains address of a  and  *p gives value stored at that address.
```

2 Write difference between (1) Array & Pointer (2) Static memory allocation & Dynamic memory allocation

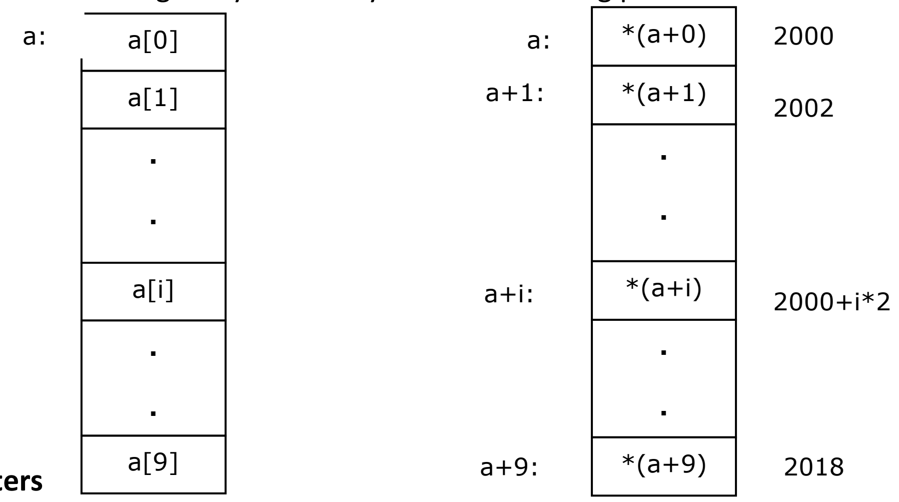
Array	Pointer
Array is a constant pointer.	Pointer variable can be changed.
It refers directly to the elements.	It refers address of the variable.
Memory allocation is in sequence.	Memory allocation is random.
Allocates the memory space which cannot resize or reassigned.	Allocated memory size can be resized.
It is a group of elements.	It is not a group of elements. It is single variable.

Static Memory allocation	Dynamic Memory allocation
A process of allocating memory at compile time is called Static memory allocation	A process of allocating memory at run time is called Dynamic memory allocation
The compiler allocates the required memory space for a declared variable	Function such as malloc() or calloc() are used to get memory dynamically
Static memory allocation stores its data in the “data segment” of the memory	Dynamic memory allocation stores its memory on heap
Memory may be wasted in static allocation.	Memory can be effectively utilized in dynamic allocation

3 Discuss relationship between array and pointer.

int a[10], *p;

- Array name is constant pointer so a is constant pointer and it always points to the first element of an array.
- a[0] is same as *(a+0), a[2] is same as *(a+2), a[i] is same as *(a+i).
- So every program written using array can always be written using pointer.



4 Explain Array of Pointers

- As we have an array of char, int, float etc..., same way we can have an array of pointers, individual elements of an array will store the address values. So, array of pointers is a collection of pointers of same type known by single name.

  Syntax :

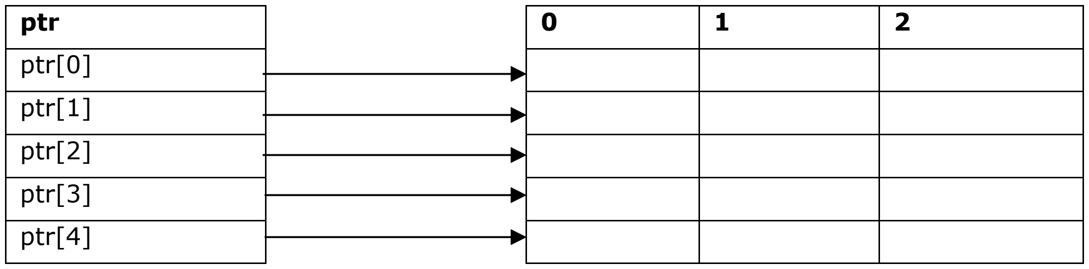
  data_type *name[size];

  Example:

  int *ptr[5];   \\Declares an array of integer pointer of size 5
  int mat[5][3]; \\Declares a two dimensional array of 5 by 2

- Now, the array of pointers ptr can be used to point to different rows of matrix as follow

```
for(i=0;i<5;i++)
{
        ptr[i]=&mat[i][0];        \\ Can be re-written as ptr[i]=mat[i];
}
```



- By using dynamic memory allocation, we do not require to declare two-dimensional array, it can be created dynamically using array of pointers.

5 Explain void pointer

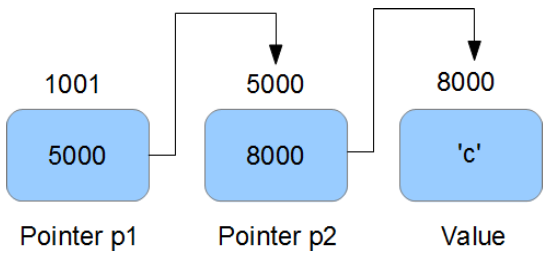
- A void pointer is pointer which has no specific data type. The keyword 'void' is preceded the pointer variable, because the data type is not specific. It is also known as a generic pointer.
- It does not have any data type associated with it. It can store address of any type of variable.
- **Declaration of void pointer;**

  `void * pointer_name;`
- **Example:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int a=10;
     float b=5.67;
     void *ptr;
     ptr = &a;
     printf("\n Value of Integer variable is %d", *((int *)ptr));
     ptr = &b;
     printf("\n Value of Integer variable is %f", *((float
*)ptr));
     getch();
     }
```

6 Explain pointer to pointer

- As the definition of pointer says that it is a special variable that can store the address of another variable. Then the other variable can very well be a pointer. This means that its perfectly legal for a pointer to be pointing to another pointer.
- Suppose we have a pointer 'p1' that points to yet another pointer 'p2' that points to a character 'c'. In memory, the three variables can be visualized as :



- So we can see that in memory, pointer p1 holds the address of pointer p2. Pointer p2 holds the address of character 'c'.
- So 'p2' is pointer to character 'c', while 'p1' is pointer to 'p2' or we can also say that 'p2' is a pointer to pointer to character 'c'.
- Now, in code 'p2' can be declared as :

```
char *p2 = &ch;
```

- But 'p1' is declared as :

```
char **p1 = &p2;
```

- **Example:**

```
#include<stdio.h>
#include<conio.h>
int main ()
{
     int var=311;
     int *ptr;
     int **pptr;
/* take the address of var */
     ptr = &var;
/* take the address of ptr using address of operator & */
     pptr = &ptr;
/* take the value using pptr */
     printf("Value of var = %d\n", var );
     printf("Value available at *ptr = %d\n", *ptr );
     printf("Value available at **pptr = %d\n", **pptr);
     getch();
     }
```

7 Dynamic Memory Allocation

- The exact size of array is unknown until the compile time.
- The size of array you have declared initially can be sometimes insufficient and sometimes more than required.
- Dynamic memory allocation allows a program to obtain memory space, while running or to release space when no space is required.
- Although, C language inherently does not has any technique to allocated memory dynamically, there are 4 library functions under "*stdlib.h*" for dynamic memory allocation.

8 Explain malloc with example

- The name malloc stands for "memory allocation". The function malloc() reserves a 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()**
  ptr = (cast-type*) malloc(byte-size)
- Here, ptr is pointer of cast-type. The malloc() function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.
- ptr = (int*) malloc(100*sizeof(int));

- **Example:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
  int n,*ptr,i;
  printf("Enter the size of an array:");
  scanf("%d",&n);
  ptr = (int *)malloc(n*sizeof(int));
  for(i=0;i<n;i++)
  {
       ptr[i]    = i+1;
  }
  for(i=0;i<n;i++)
  {
       printf("\n\t%d",ptr[i]);
  }
  free(ptr);
  getch();
}
```

9 Explain calloc with example

- The name calloc stands for "contiguous allocation". The only difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size.

- **Syntax of calloc()**
  ptr = (cast type *)calloc(n,element-size);
- This statement will allocate contiguous space in memory for an array of n elements. For example:
- ptr= (int*) calloc((25,sizeof(int));
- **Example:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
   int n,*ptr,i;
   printf("Enter the size of an array:");
   scanf("%d",&n);
   ptr = (int *)calloc(n,sizeof(int));
   for(i=0;i<n;i++)
   {
         ptr[i]    = i+1;
   }
   for(i=0;i<n;i++)
   {
    printf("\n\t%d",ptr[i]);
   }
   free(ptr);
   getch();
}
```

10 Explain realloc with example

- If the previously allocated memory is insufficient or more than sufficient. Then, you can change memory size previously allocated using realloc().
- **Syntax of realloc():**
  ptr= realloc(ptr,newsize);
- Here, ptr is reallocated with size of newsize.
- **Example:**

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
        int *ptr,i,n1,n2;
        printf("Enter size of array: ");
        scanf("%d",&n1);
        ptr=(int*)malloc(n1*sizeof(int));
        printf("Address of previously allocated memory: ");
        for(i=0;i<n1;++i)
        printf("%u\t",ptr+i);
```

```
          printf("\nEnter new size of array: ");
          scanf("%d",&n2);
          ptr=realloc(ptr,n2); for(i=0;i<n2;++i)
          printf("%u\t",ptr+i);
          getch();
 }
```

11 Swap value of two variables using pointer. OR
Swap value of two variables using call by reference

```
#include<stdio.h>
void swap(int *,int *);
void main()
{
     int a,b;
     printf("Enter two numbers:");
     scanf("%d%d", &a, &b);
     swap(&a, &b);
     printf("a=%d b=%d", a, b);
     getch();
}
void swap(int *a, int *b)
{
     int temp;
     temp=*a;
     *a=*b;
     *b=temp;
}
```

12 Write a C program to calculate sum of 10 elements of an array using pointers

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int *p, a[10], sum=0, i;
     p=a;
     printf("Enter elements:");
     for(i=0; i<10; i++)
     {
          scanf("%d", &a[i]);
     }
     for(i=0; i<10; i++)
     {
          sum=sum+*p;
          p++;
     }
     printf("sum=%d", sum);
     getch();
}
```

13 Advantages and Disadvantages of pointer

- **Advantages:**

✓ Pointers provide direct access to memory.
✓ Reduces the execution time of the program
✓ Pointer save memory space.
✓ Pointers provide a way to return more than one value to the functions.
✓ Reduces the storage space and complexity of the program
✓ Provides an alternate way to access array elements
✓ Pointers helps us to build complex data structures like linked list, stack, queues, trees, graphs etc.
✓ Pointers allow us to resize the dynamically allocated memory block.

- **Disadvantage of Pointer**

✓ If pointers are updated with incorrect values, it might lead to memory corruption.
✓ Uninitialized pointers might cause segmentation fault.