Compiled by ab.

Computer Concept & Programming

I semester, BScCSIT

Ambition Guru College

Compiled by :
Ankit Bhattarai

Unit 6

(5 hrs.)

Pointers

- Introduction, Pointer Declaration, Initializing Pointers, Arithmetic Operations with Pointer
- Pointers and Arrays, Pointers and Strings, Pointers and Functions, Pointer to a Pointer, Pointer to Void, Dynamic Memory Allocation.

Concept of Pointer

- Pointer is a variable that stores/points the address of another variable. Pointer variable can only contains the address of a variable of the same data type.

eg.

```
int *ptr; //pointer declaration
int a=5;
ptr=&a; //address of variable a is assigning to pointer variable ptr
```

Here `ptr` is a pointer variable that only contains the address of integer data type.

Pointer Declaration

- Pointer variable is declared with an asterisk (*) operator before a variable name. This operator is called pointer/indirection or dereference operator.

Syntax: `data_type *pointer_variable_name;`

Data type of the pointer must be same as the data type of the variable to which the pointer variable is pointing.

eg. `int *ptr;`

Declares the variable `ptr` which is a pointer variable that points to an integer data type.

Initialization of Pointers

- Pointer initialization is the process of assigning address of variable to a pointer variable .In C programming language ampersand (&) operator is used to determine the address of variable. The & (immediately preceding a variable name) returns the address of variable associated with it.
- Pointer variable always points to variables of same data type.

```
int main()
{
int a=10;
int *ptr; //pointer declaration
ptr=&a; //pointer initialization
}
```

Here, pointer variable ptr of integer type is pointing the address of integer variable a.

Initialization of Pointers

Once, address of variable is assigned to pointer, then to access the value of variable, pointer is dereferenced using indirection/dereference operator (*).

Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a=5;
int *ptr; //declaration of pointer variable
ptr=&a; //initializing the pointer
printf("\n%d",a); //prints the value of a
printf("\n%d",*ptr); //prints the value of a
printf("\n%d",*(&a)); //prints the value of a
printf("\n%u",&a); //prints the address of a
printf("\n%u",ptr); //prints the address of a
printf("\n%u",&ptr); //prints the address of pointer variable ptr
getch();
return 0;
}
```

Some Definitions:

- **Bad Pointer:** When a pointer variable is declared and if any valid address is not assigned to it then pointer is known as bad pointer.
- A dereference operation on a bad pointer is a serious runtime error. Each pointer must be assigned a valid address before it can support dereference operations. Before that pointer is bad and must not be used.
- It is always a best practice to initialize a pointer NULL values, when they are declared and check for whether the pointer is NULL pointer when using the pointer.

Some Definitions:

- **Void pointer** is a special type of pointer that can point any data type (int or float or char or double).

Declaration: void *pointer_name;

Example:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a=10;
float b=4.5;
void *ptr;
ptr=&a;
printf("a=%d",*(int*)ptr);
ptr=&b;
printf("b=%f",*(float*)ptr);
getch();
return 0;
}
```

Output:

```
a=10
b=4.500000
```

Some Definitions: Null Pointer

- A Null pointer is a pointer which is pointing to nothing.
- Pointer which is initialized with NULL value is considered as NULL pointer.
- We can define a null pointer using a predefined constant NULL, which is defined in header files stdio.h , stdef.h, stdlib.h.

Some of the most common use cases for NULL are

i. To initialize a pointer variable when that pointer variable isn't assigned any valid memory address yet.

```
int * ptr = NULL;
```

ii. To check for null pointer before accessing any pointer variable. By doing so, we can perform error handling in pointer related code e.g. dereference pointer variable only if it's not NULL.

```
if(ptr != NULL) /*We could use if (ptr) as well*/
{
/*Some code*/
}
else
{
/*Some code*/
}
```

Pointer Arithmetic

Pointer Arithmetic

As pointer holds the memory address of variable, some arithmetic operations can be performed with pointers. They are as follows.

- Increment
- Decrement
- Addition
- Subtraction

If arithmetic operations done values will be incremented or decremented as per data type chosen. Let `ptr=&arr[0]=1000` i.e Base address of array.

int type pointer (2-byte space)	**float type pointer (4-byte space)**	**char type pointer (1-byte pointer)**
ptr++=ptr+1=1000+2= 1002 is the address of next element.	ptr++=ptr+1=1000+4= 1004 is the address of next element	ptr++=ptr+1=1000+1= 1001 is the address of next element
ptr=ptr+4=1000+(2*4)=1008 ie. Address of 5th integer type element (&arr[4])	ptr=ptr+4=1000+(4*4)=1016 ie. Address of 5th float type element (&arr[4])	ptr=ptr+4=1000+(1*4)=1004 ie. Address of 5th char type element(&arr[4])

Note: Similar operation can be done for decrement

Pointer Arithmetic

- Note:

ptr	Pointer to first row
ptr+i	Pointer to ith row
*(ptr+i)	Pointer to first element in the ith row
*(ptr+i)+j	Pointer to the jth element in the ith row
((ptr+i)+j)	Value stored in the cell(i,j) (ith row and jth column)

Pointer Arithmetic

- **Example for pointer increment/decrement**

```
#include<stdio.h>
#include<conio.h>
int main()
{
 int arr[5] = {10,20,30,40,50};
 int *ptr1,*ptr2;
 ptr1=&arr[0];
 ptr1++;
 printf("\nvalue %d has address %u",*ptr1,ptr1);
 ptr2=&arr[4];
 ptr2-- ;
 printf("\nvalue %d has address %u",*ptr2,ptr2);
 getch();
 return 0;
}
```

Output:

value 20 has address 7274120
value 40 has address 7274128

Pointer Arithmetic

- **Example for pointer addition/subtraction with integer constant**

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[5] = {10,20,30,40,50};
int *ptr1,*ptr2;
ptr1=&arr[0];
ptr1=ptr1+4;
printf("\nvalue %d has address %u",*ptr1,ptr1); //points to
5th element

ptr2=&arr[4];
ptr2=ptr2-4 ;

printf("\nvalue %d has address %u",*ptr2,ptr2); //points to
1st element
getch();
return 0;
}
```

Output:

value 50 has address 6487588
value 10 has address 6487572

Pointer Arithmetic

- **Other operations**

1) A pointer variable can be assigned the address of an ordinary variable.

```
int main()
{
int a=5;
int *ptr;
ptr=&a;
printf("value of a=%d",a);
printf("Address of a=%u",ptr);
return 0;
}
```

Pointer Arithmetic

- **Other operations**

2) Content of one pointer variable can be assigned to other pointer variable provided they point to same data type.

```
int main()
{
int arr[5]={1,2,3,4,5};
int *ptr1,*ptr2;
ptr1=&arr[0];
ptr2=ptr1; //assign element of ptr1 to ptr2
printf("ptr1 contains %u and ptr2 contains %u",ptr1,ptr2);
return 0;
}
```

Pointer Arithmetic

- **Other operations**

3) One pointer variable can be subtracted from another provided that both variables points to the element of same array.

```
int main()
{
int arr[5]={1,2,3,4,5};
int *ptr1,*ptr2;
ptr1=arr;
ptr2=&arr[4];
printf("Difference of two pointer =%d",ptr2-ptr1);
return 0;
}
```

Pointer Arithmetic

- **Other operations**

4) Two pointers variables can be compared provided both pointers points to object of same data type.

```
int main()
{
int arr[5]={1,2,3,4,5};
int *ptr1,*ptr2;
ptr1=&arr[0];
ptr2=&arr[4];
if(ptr2>ptr1)
printf("ptr2 is far from ptr1");
else
printf("ptr1 is far from ptr2");
return 0;
}
```

Pointer Arithmetic

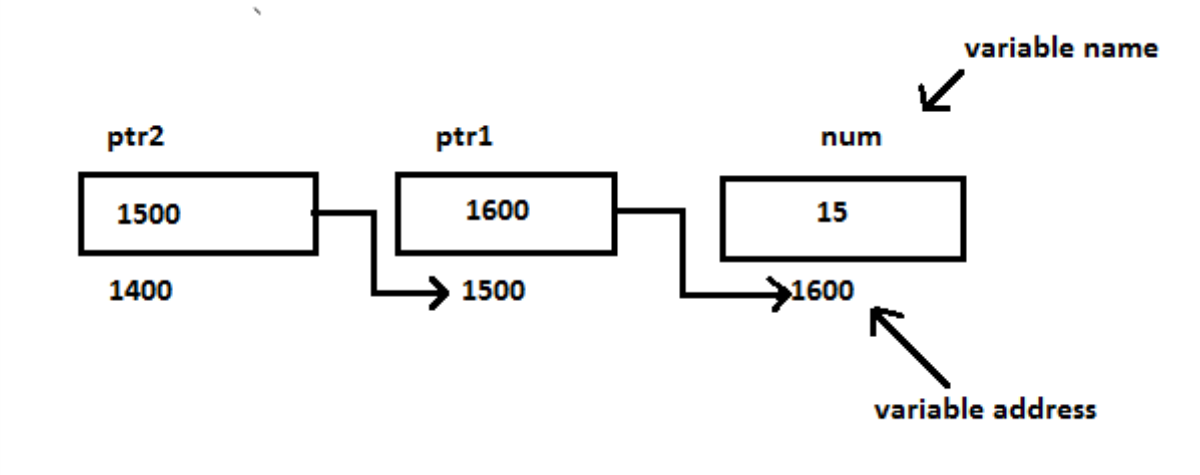
Invalid pointer operations

- Addition of two pointer (ptr1+ptr2)
- Multiplication of two pointer(ptr1*ptr2)
- Addition or subtraction of float or double data type to or from a pointer
- Multiplication of pointer with constant (ptr1*5)
- Division of two pointer or with constant (ptr1/5)

Double pointer/chain of pointer /pointer to pointer/ Double indirection

Double pointer/chain of pointer /pointer to pointer/ Double indirection

- When pointer holds the address of another pointer then such type of pointer is called double pointer /chain of pointer.



- As per diagram, here ptr1 is a normal pointer that holds the address of an integer variable num. There is another pointer ptr2 in the diagram that holds the address of another pointer ptr1, the pointer ptr2 here is pointer to pointer(or double pointer).
- A variable that is pointer to pointer must declared using two indirection operator symbol in front of name.eg. int **ptr2;

Double pointer/chain of pointer /pointer to pointer/ Double indirection

- Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num=15;
int *ptr1;
int **ptr2;
ptr1=&num;
ptr2=&ptr1;
printf("num=%d\n",*ptr1);
printf("num=%d",**ptr2);
getch();
return 0;
}
```

Output
num= 15
num= 15

Passing pointer to function

Passing pointer to function

- Pointers can be passed to functions as arguments.
- When we pass a pointer, the address of the variable is sent, not the value.
- This allows the function to access and modify the original variable.
- Any changes made through the pointer affect the actual variable.
- This method is known as Call by Reference.

It is useful when:

✓ We want the function to change original values.

✓ We want to return multiple values from a function.

Passing pointer to function

- Program to illustrate passing pointer to function.

```
#include<stdio.h>
#include<conio.h>
void addGraceMarks(int *m);
int main()
{
int marks;
printf("Enter the actual marks\n");
scanf("%d",&marks);
addGraceMarks(&marks);
printf("The final marks is:%d",marks);
getch();
return 0;
}
void addGraceMarks(int *m)
{
   *m=*m+10;
}
```

Passing pointer to function

- Program to convert uppercase letter into lower and vice versa passing pointer to function.

```
#include<stdio.h>
#include<conio.h>
void conversion(char *);
int main()
{
char ch;
printf("Enter the character\n");
scanf("%c",&ch);
conversion(&ch);
printf("The corresponding character is:%c",ch);
getch();
return 0;
}

void conversion(char *c)
{
   if(*c>='a'&&*c<='z')
      {
      *c=*c-32;
      }
   else if (*c>='A'&&*c<='Z')
      {
      *c=*c+32;
      }
}
```

Returning multiple values from function

Returning multiple values from function

- Can function return more than one value? Justify your answer with examples,
- Does function return single or multiple value? When and how a function will return single or multiple value? Illustrate with examples.
- How can function return multiple values? Explain with example?

- Normally, functions in C cannot return more than one value when arguments are passed by value.
- By using pointers, we can effectively return multiple values from a function.
- This is done by passing the address of variables to the function.
- Inside the function, we can modify the values at those addresses using pointers.
- These changes are reflected in the original variables (actual arguments).
- This technique is known as Call by Reference.
- We can pass any number of variables as pointers to return multiple results

Returning multiple values from function

- Example:

```
#include<stdio.h>
#include<conio.h>
void areaperi(int r,float *area,float *peri);
int main()
{
int rad;
float a,p;
printf("Enter the radious\n");
scanf("%d",&rad);
areaperi(rad,&a,&p);
printf("Area of circle=%f",a);
printf("Perimeter of circle=%f",p);
getch();
return 0;
}

void areaperi(int r,float *area,float *peri)
{
*area=3.14*r*r;
*peri=2*3.14*r;
}
```

Returning multiple values from function

Example:

```
#include<stdio.h>
#include<conio.h>
void areaperi(int r,float *area,float *peri);
int main()
{
int rad;
float a,p;
printf("Enter the radious\n");
scanf("%d",&rad);
areaperi(rad,&a,&p);
printf("Area of circle=%f",a);
printf("Perimeter of circle=%f",p);
getch();
return 0;
}

void areaperi(int r,float *area,float *peri)
{
  *area=3.14*r*r;
  *peri=2*3.14*r;
}
```

Advantages of Pointer

Advantages of Pointer

- Pointers enable dynamic memory allocation (DMA), allowing efficient use of memory at runtime.
- Pointers simplify array manipulation, enabling fast traversal and element access through arithmetic.
- Pointers allow efficient handling of structures, especially when passing large data to functions without copying.
- Pointers enable function arguments to be passed by reference, allowing in-place modification of variables.
- Pointers provide flexibility and efficiency by directly accessing and managing memory addresses.

Array and pointer

1D array with pointer

Index	0	1	2	3	4
Value	5	10	15	20	25
Address	1000	1002	1004	1006	1008

```
int arr[5]={5,10,15,20,25};
int *ptr, i;
ptr=arr; //similar to ptr=&arr[0]
By assuming array starts at location 1000
```

&arr[0]=1000	arr[0]=5	ptr+0=1000	*(ptr+0)=5
&arr[1]=1002	arr[1]=10	ptr+1=1002	*(ptr+1)=10
&arr[2]=1004	arr[2]=15	ptr+2=1004	*(ptr+2)=15
&arr[3]=1006	arr[3]=20	ptr+3=1006	*(ptr+3)=20
&arr[4]=1008	arr[4]=25	ptr+4=1008	*(ptr+4)=25

1D array with pointer

Index	0	1	2	3	4
Value	5	10	15	20	25
Address	1000	1002	1004	1006	1008

Accessing value	Accessing Address

printf("%d",*(ptr+i)); //displays the arr[i] value
printf("%d",*ptr); //display the arr[0] value
printf("%d",*(arr+i); //displays the arr[i] value
	printf("%u",(ptr+i)); //displays the address of arr[i]
	printf("%u", ptr); //display the address of arr[i]
	printf("%u",(arr+i)); //displays the address of arr[i]

1D array with pointer

WAP to input n number in array and display it using pointer.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[100],i,n,*ptr;
ptr=arr;
printf("Enter the number of elements you want
to enter");
scanf("%d",&n);

printf("\nEnter %d elements",n);
for(i=0;i<n;i++)
{
   scanf("%d",(ptr+i));
}

printf("Entered elements are\n");
for(i=0;i<n;i++)
{
   printf("%d",*(ptr+i));
}
getch();
return 0;
}
```

1D array with pointer

WAP to input n number in array and find sum of all elements using pointer

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[100],n,i,*ptr,sum=0;
ptr=arr;
printf("Enter number of elements you want to
enter");
scanf("%d",&n);

printf("Enter %d elements",n) ;
for(i=0;i<n;i++)
{
   scanf("%d",(ptr+i));
}

for(i=0;i<n;i++)
{
   sum=sum+*(ptr+i);
}
printf("The sum of all elements is %d",sum);
getch();
return 0;
}
```

1D array with pointer

Write a program to sort n integer values in an array using pointer.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[100],n,i,j,temp,*ptr;
ptr=arr;
printf("Enter number of elements you want to
enter");
scanf("%d",&n);
printf("Enter %d elements",n) ;
for(i=0;i<n;i++)
{
   scanf("%d",(ptr+i));
}

for(i=0;i<n-1;i++)
  {
    for(j=0;j<n-i-1;j++)
      {
        if(*(ptr+j)>*((ptr+j)+1))
         {
          temp=*(ptr+j);
          *(ptr+j)=*((ptr+j)+1);
          *((ptr+j)+1)=temp;
         }
      }
}
```

1D array with pointer

Write a program to sort n integer values in an array using pointer.

```
printf("\nThe sorted array are");
for(i=0;i<n;i++)
{
   printf("\n%d",*(ptr+i));
}
getch();
return 0;
}
```

1D array with pointer

Write a program using pointers to read in an array of integers and prints its elements in reverse order.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[100],i,n,*ptr;
ptr=arr;
printf("Enter number of elements you want to enter");
scanf("%d",&n);

printf("Enter %d elements",n) ;
for(i=0;i<n;i++)
{
  scanf("%d",(ptr+i));
}

printf("\nThe array elements are");
for(i=0;i<n;i++)
{
  printf("\n%d",*(ptr+i));
}

printf("\nThe array elements in reverse order are");
for(i=n-1;i>=0;i--)
{
  printf("\n%d",*(ptr+i));
}
getch();
return 0;
}
```

Array of pointers

Array of pointers

- Array of pointers is a collection of address.
- The address present in the array of pointers can be address of variable or address of array element.

Array of pointers can be declared as:

```
datatype *pointer_name[size];
```

eg.int *ptr[4];

This statement declares an array of 4 pointers, each of which points to an integer value. The first pointer is called ptr[0] ,the second is ptr[1] ,third is ptr[2] and fourth is ptr[3].

Array of pointers

Example

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int a=5,b=6,c=7,d=8,i;
   int *ptr[4];
   ptr[0]=&a;
   ptr[1]=&b;
   ptr[2]=&c;
   ptr[3]=&d;

for(i=0;i<4;i++)
{
  printf("%d",*ptr[i]);
}
getch();
return 0;
}
```

Array of pointers

WAP to input 5 integer numbers and add all elements using array of pointer.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int *ptr[5];
int arr[5],i,sum=0;

for(i=0;i<5;i++)
{
   ptr[i]=&arr[i];
}
printf("Enter 5 elements\n");

for(i=0;i<5;i++)
{
   scanf("%d",&arr[i]);
}

for(i=0;i<5;i++)
{
   sum=sum+*ptr[i];
}
printf("sum=%d",sum);
getch();
return 0;
}
```

2D array using pointer

2D array using pointer

Syntax:

```
datatype(*ptr_variable)[size2];
```

Example:

Let us suppose arr is a 2-D integer array having 3 rows and 4 columns, we can declare arr as

```
int(*ptr)[4];
```

```
#include<stdio.h>
#include<conio.h>
int main()
{
int arr[3][4] ={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
int i, j;
int (*ptr)[4];
ptr = arr; //&arr[0][0]
for(i = 0; i < 3; i++)
{
   for(j = 0; j < 4; j++)
   {
    printf("\narr[%d][%d]=%d", i, j, *( *(ptr + i) + j) );
   }
printf("\n");
}
getch();
return 0;
}
```

Note: As array name holds the base address (i.e. address of first element of array).Then, if arr is a 2D array we can access any element arr[i][j] of the array using the pointer expression.
((arr+i)+j)

2D array using pointer

WAP to add two matrix of order m*n by using the concept of pointer.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a[20][20],b[20][20],c[20][20],i,j,m,n;
printf("Enter the value of m & n:");
scanf("%d%d",&m,&n);
printf("Enter the first matrix:");
for(i=0;i<m;i++)
{
   for(j=0;j<n;j++)
   {
     scanf("%d",(*(a+i)+j));
   }
}

printf("\nEnter second matrix:");
for(i=0;i<m;i++)
{
   for(j=0;j<n;j++)
   {
     scanf("%d",(*(b+i)+j));
   }
}
```

2D array using pointer

WAP to add two matrix of order m*n by using the concept of pointer.

```
printf("\nAddition of two Matrix:\n");
for(i=0;i<m;i++)
{
  for(j=0;j<n;j++)
  {
    *(*(c+i)+j)=*(*(a+i)+j)+ *(*(b+i)+j);
    printf("%d ",*(*(c+i)+j));
  }
  printf("\n");
}
getch();
return 0;
}
```

Array Vs Pointer

Question: Differences between array and pointer.

Arrays	Pointers
An array is a collection of elements of the same type stored in contiguous memory locations.	A pointer is a variable that stores the memory address of another variable.
Elements are accessed using the array index, e.g., array[index]	A pointer accesses elements it points to by dereferencing, e.g., *pointer
The size of an array is fixed and defined at the time of declaration	A pointer is of a fixed size (depends on the architecture, usually 4 or 8 bytes), but it can point to blocks of memory of any size.
Directly access data.	Provides an extra level of indirection to access data.
Limited to index manipulation within the array bounds.	Pointers can be reassigned to point to different memory addresses.
Declared using the syntax: *data_type name[size];*	Declared using the syntax: *data_type *name;*

Dynamic Memory Allocation

Dynamic Memory Allocation

- The process of allocating and releasing memory at runtime is known as Dynamic memory allocation.
- Using DMA memory space is reserved during program execution and release space when no longer required.

Functions used in Dynamic Memory allocation

- There are four library functions: malloc(),calloc(), free() and realloc() are for dynamic memory management.

Dynamic Memory Allocation

1) malloc()

- The malloc() function reserves a block of memory of specified size and returns a pointer of type void, which can be casted into pointer of any form.
- The memory allocation can fail if the space in heap is not sufficient to satisfy the request. If it fails it returns NULL.

Syntax:

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

where, ptr is pointer of type cast-type.

eg.

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

A memory space equivalent to “100 times the size of integer” is reserved and address of the first memory allocated is assigned to pointer ptr of type int.

Dynamic Memory Allocation

2) calloc()

calloc() allocates multiple block of storage, each of same size. All bytes are initializes to zero and pointer to the first byte of allocated region is returned. If there is not enough space, a NULL pointer is returned.

Syntax:

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

e.g.

```
int *ptr;
ptr=(int*) calloc(25,sizeof(int));
```

This statement allocates contiguous space in memory for 25 elements each with size of int.

Dynamic Memory Allocation

3) realloc()

This function is used to modify the size of previously allocated space.

Syntax:

ptr=realloc(ptr,newsize); //Reallocation of space

- *Sometimes previously allocated space is not sufficient we need to additional space and sometimes allocated memory is much larger than necessary. In both situation, we can change the memory size already allocated with the help of function realloc().*
- *This function allocates the new memory space of size newsize to the pointer variable ptr and returns a pointer to the first byte of new memory.*

Dynamic Memory Allocation

4) free()

It releases the previously allocated space by malloc(), calloc(), and the realloc() function.

Its syntax:

free(ptr);

- *The memory dynamically allocated is not returned to the system until the programmer returns the memory explicitly. This can be done by using free() fuction. This function is used to release the space when it is not required.*
- *Where ptr is a pointer to memory block which has already been created by malloc(),calloc() or realloc() function.*

DMA

Program to dynamically allocate memory for the array elements using calloc() function and read and display the array elements.

Perform similar operations using malloc()

Hint:

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

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int *ptr;
    int i,n;
    printf("Enter number of elements to be entered\n");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));
    if(ptr == NULL)
    {
    printf("Error! memory not allocated.");
    exit(0);
    }
    printf("Enter %d numbers\n",n);
    for(i=0;i<n;i++)
    {
      scanf("%d",(ptr+i));
    }
   printf("The elments are\n");
   for(i=0;i<n;i++)
   {
        printf("\n%d",*(ptr+i));
   }
  free(ptr);
  getch();
  return 0;
}
```

DMA

Program to illustrate the use of realloc() function

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int *ptr = (int *)malloc(sizeof(int)*2);
int i;
int *ptrnew;
*ptr = 10;
*(ptr + 1) = 20;
ptrnew = (int *)realloc(ptr, sizeof(int)*3);
*(ptrnew + 2) = 30;
for(i = 0; i < 3; i++)
{
   printf("%d\t ", *(ptrnew + i));
}
free(ptr);
free(ptrnew);
getch();
return 0;
}
```

Output:

10 20 30

DMA

Write a program to find the sum of 5 numbers supplied by users using DMA.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int *ptr;
int i,n,sum=0;

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

if(ptr == NULL)
{
printf("Error! memory not allocated.");
exit(0);
}

printf("Enter 5 numbers\n",n);
for(i=0;i<5;i++)
{
   scanf("%d",(ptr+i));
}
for(i=0;i<5;i++)
{
    sum=sum+*(ptr+i);
}
printf("The sum of 5 numbers is %d",sum);
free(ptr);
getch();
return 0;
}
```

DMA

WAP to find the highest and lowest element of an array using DMA

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int *ptr;
int i,n,large,small;
printf("Enter number of elements to be entered\n");
scanf("%d",&n);
ptr=(int*)calloc(n,sizeof(int));
if(ptr == NULL)
{
  printf("Error! memory not allocated.");
  exit(0);
}
printf("Enter %d numbers\n",n);
for(i=0;i<n;i++)
{
  scanf("%d",(ptr+i));
}
```

DMA

WAP to find the highest and lowest element of an array using DMA

```
printf("Entered array elements are\n");
for(i=0;i<n;i++)
{
printf("\n%d",*(ptr+i));
}
large=*ptr;
small=*ptr;
for(i=0;i<n;i++)
{
   if(large<*(ptr+i))
      large=*(ptr+i);
   if(small>*(ptr+i))
      small=*(ptr+i);
}
printf("largest element =%d",large);
printf("smallest element =%d",small);
free(ptr);
getch();
return 0;
}
```

Questions:

1. Explain the concept of pointers in C/C++. How are pointers declared and initialized? Write a program to demonstrate pointer declaration and initialization.
2. What are arithmetic operations on pointers? Describe different pointer arithmetic operations with examples.
3. How are pointers used with arrays? Write a program to demonstrate accessing array elements using pointers. Explain the output.
4. Discuss how pointers are used with strings in C/C++. Write a program to print a string using a pointer.
5. Explain how pointers are passed to and returned from functions. Write a program to swap two numbers using pointers and functions.
6. What is a pointer to a pointer? Explain with the help of a program.
7. Explain the use of void pointers in C/C++. Why are they called generic pointers? Write a simple program to illustrate their use.
8. What is dynamic memory allocation? Explain malloc(), calloc(), realloc(), and free() functions with example code.
9. Write a program to dynamically allocate memory for an array using malloc() and calculate the average of the elements. Explain how memory is allocated and released.
10. Compare and contrast static memory allocation and dynamic memory allocation. Discuss with examples.

Compiled by ab.

THANK YOU

Any Queries ?