C Programming

COMMENTS

- Comments are specially marked lines of text in the program that are not evaluated
  - Single line of comment:
    - `// comment here`
  - Multiple line Comments:

    -
      ```
      /* comment here

       line two */
      ```

```
// Author: xyz
#include <stdio.h>

/* main() function, where program
     execution starts */
int main()
{
      int i; // used for roll no
      Return 0;
}
```

Multi-way Decision Statement

```
if(condition_1)
   statement_1;
else if (condition_2)
   statement_2;
else if(condition_3)
   statement_3;
else
   statement_4;
next_statement;
```

```
#include<stdio.h>
int main()
{
     int day;
     printf(“\n Enter any number from 1 to 7 :
“);
     scanf(“%d”,&day);

     if(day==1){
          printf(“\n SUNDAY”);
     }else if(day==2){
          printf(“\n MONDAY”);
     }else if(day==3){
          printf(“\n TUESDAY”);
     }else if(day==4){
          printf(“\n WEDNESDAY”);
     }else if(day==5){
          printf(“\n THURSDAY”);
     }else if(day==6){
          printf(“\n FRIDAY”);
     }else if(day==7){
          printf(“\n SATURDAY”);
     }else{
          printf(“\n Wrong Number”);
     }
     return 0;
}
```

Switch Case

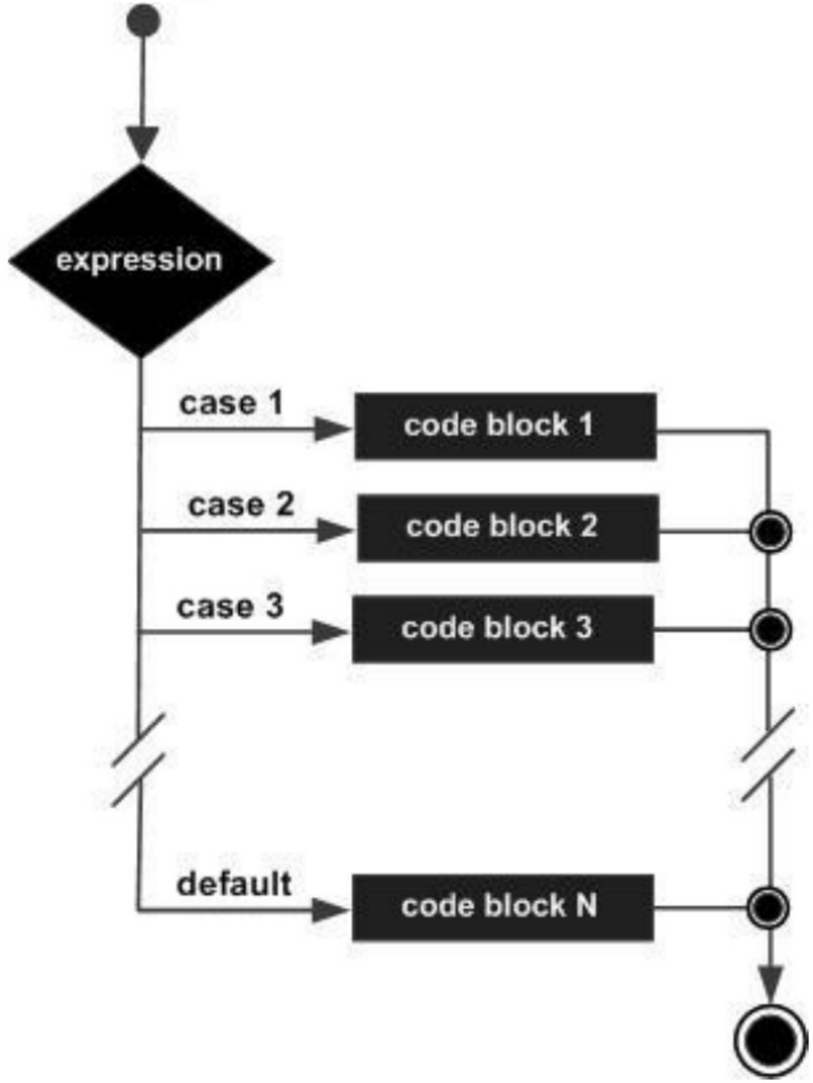
- A switch case statement is a **multi-way decision statement**.
- It is used when there is **only one variable to evaluate** in the expression
- test condition only **use integer (or character)** constants.

```
switch(<expression>)
{
   case  <Value_1>  :
         statement(s);
         break;
   case  <Value_2>  :
         statement(s);
         break;
   case  <Value_3>  :
         statement(s);
         break;
   …
   …
   case  <Value_n> :
         statement(s);
         break;
   default :
         statement(s);
}
```

```
#include<stdio.h>
int main()
{
     int day;
     printf(“\n Enter any number from 1 to 7 : “);
     scanf(“%d”,&day);
     switch(day)
     {
          case 1:
                printf(“\n SUNDAY”);
                break;
          case 2 :
                printf(“\n MONDAY”);
                break;
          case 3 :
                printf(“\n TUESDAY”);
                break;
          case 4 :
                printf(“\n WEDNESDAY”);
                break;
          case 5 :
                printf(“\n THURSDAY”);
                break;
          case 6 :
                printf(“\n FRIDAY”);
                break;
          case 7 :
                printf(“\n SATURDAY”);
                break;
          default:
                printf(“\n Wrong Number”);
     }
     return 0;
}
```

Switch Case

Flow Diagram



```
#include <stdio.h>

int main() {
     int number;
     Scanf(“%d”,&number);
     switch (number) {
          case 1:
          case 2:
          case 3:
               printf("One, Two, or Three.\n");
               break;
          case 4:
          case 5:
          case 6:
               printf("Four, Five, or Six.\n");
               break;
          default:
               printf("Greater than Six.\n");
     }
}
```

Array

Can you imagine how long we have to write the declaration part by using normal variable declaration?

```
int main(void)
{
   int mark1, mark2, mark3, mark4, …, …, mark998,
   stuMark999, mark1000;
   scanf(“%d”, &mark1);
   scanf(“%d”, &mark2);
   …
   printf(“%d”, mark1);
   printf(“%d”, mark1);

   …
   return 0;
}
```

We cannot also use any loop
For mark1, mark2, …
Because each variable
has different name

Array

int mark[10];

1st element	2nd element	3rd element	4th element	5th element	6th element	7th element	8th element	9th element	10th element
mark[0]	mark[1]	mark[2]	mark[3]	mark[4]	mark[5]	mark[6]	mark[7]	mark[8]	mark[9]

```
int main(void)
{
    int mark[10];
    scanf(“%d”, &mark[0]);
    scanf(“%d”, &mark[1]);
    …
    printf(“%d”, mark[0]);
    printf(“%d”, mark[1]);

    …
    }
```

```
int main(void)
{
    int mark[10];
    int I;
    for(i=0;i<10;i++){
        scanf(“%d”, &mark[i]);
    }

    for(i=0;i<10;i++){
        printf(“%d”, mark[i]);
    }
    …
    }
```

Array

- An array is a collection of similar data elements.
- The elements of the array are stored in consecutive memory locations and are referenced by an index starts with **0**

99	67	78	56	88	90	34	85
mark[0]	mark[1]	mark[2]	mark[3]	mark[4]	mark[5]	mark[6]	mark[7]
1000	1002	1004	1006	1008	1010	1012	1014

int mark[8];

- Declaration of an array mainly contains three things:
  - data type : **int**
  - Name of the : **mark**
  - Number of elements : **8** (index : 0 to 7)

Array

```
int main(void)
{
        int mark[5];
        int i;
        for(i=0 ; i<5 ; i++){
            scanf(“%d”, &mark[i]);
        }

        for(i=0 ; i<5 ; i++){
            printf(“%d”, mark[i]);
        }
        return 0;
}
```

```
int main(void)
{
        int mark[5]={2,4,5,12,3};
        int i;

        for(i=0 ; i<5 ; i++){
            printf(“%d”, mark[i]);
        }
        return 0;
}
```

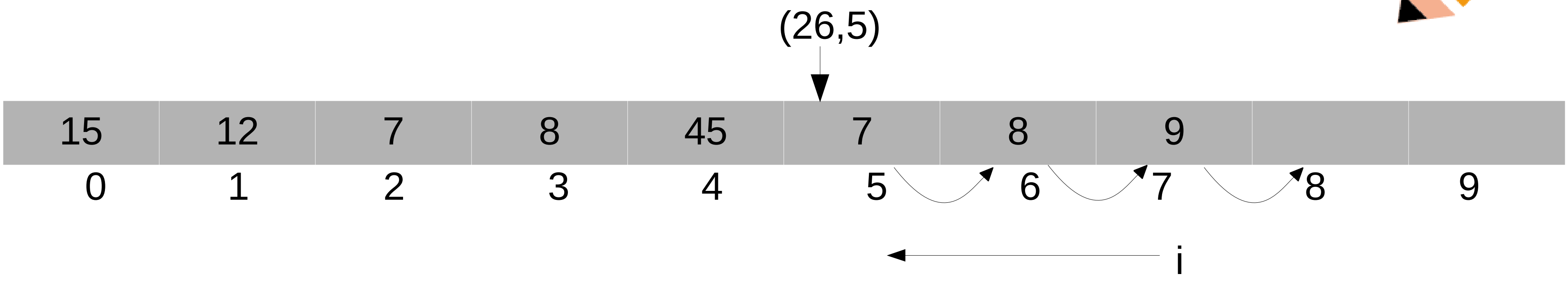
```
int main(void)
{
        int mark[5];
        int I, sum=0;
        for(i=0 ; i<5 ; i++){
            scanf(“%d”, &mark[i]);
        }
        for(i=0 ; i<5 ; i++){
            sum=sum+mark[i];
        }
        printf(“%d”, sum);
        return 0;
}
```

```
int main(void)
{
        int mark[5]={2,4,5,12,3};
        int I, sum=0;

        for(i=0 ; i<5 ; i++){
            sum=sum+mark[i];
        }
        printf(“%d”, sum);
        return 0;
}
```

Problems

- Display the array in reverse order.
- Find the smallest element in an array.
- Write a program to interchange the biggest and the smallest number in the array.
- Write a program to find the mean of n numbers using arrays.
- Write a program to search an element in an array.
- Write a program to merge two integer arrays. Also display the merged array in reverse order.
- Write a program to insert a number in an array.(either in sorted array or in an unsorted array by index)
- Write a program to delete a number from an array.

Inserting an Element

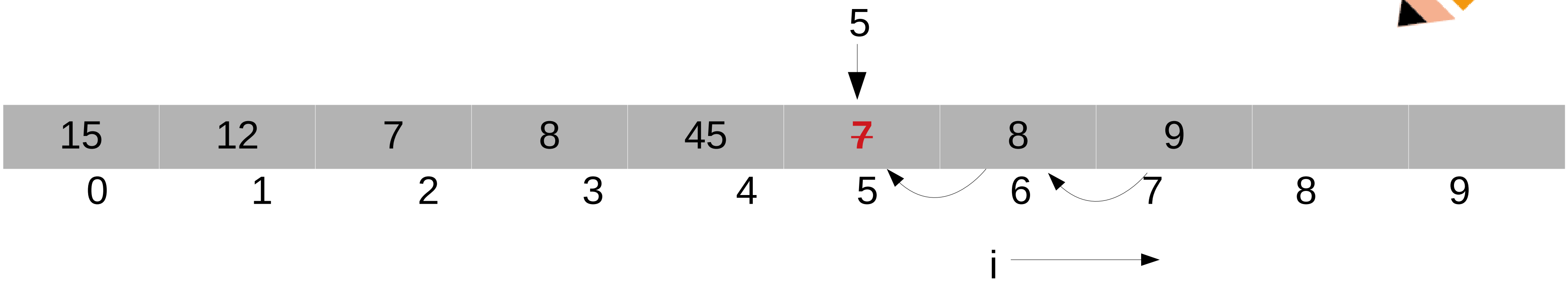


(26,5)

15	12	7	8	45	7	8	9		
0	1	2	3	4	5	6	7	8	9

15	12	7	8	45	26	7	8	9	
0	1	2	3	4	5	6	7	8	9

Deleting an Element



15	12	7	8	45	~~7~~	8	9		
0	1	2	3	4	5	6	7	8	9

15	12	7	8	45	8	9			
0	1	2	3	4	5	6	7	8	9

Array & Function

```
int main(void)
{
        int mark[5]={12, 34, 35, 14, 57};
        int i;
        for(i=0 ; i<5 ; i++){
            printf(“%d”, mark[i]);
        }
        return 0;
}
```

```
void display(int data[], int size){
        int i;
        for(i=0 ; i<size ; i++){
            printf(“%d”, data[i]);
        }
}
int main(void) {
        int mark[5]={12, 34, 35, 14, 57};
        display(mark,5);
        return 0;
}
```

```
int main(void)
{
        int mark[5]={12, 34, 35, 14, 57};
        int I,sum=0;
        for(i=0 ; i<5 ; i++){
            sum=sum+mark[i];
        }
        printf(“Total=%d”, sum);
        return 0;
}
```

```
int total(int data[], int size){
        int i,sum=0;
        for(i=0 ; i<5 ; i++){
            sum=sum + data[i];
        }
        return sum;
}
int main(void) {
        int mark[5]={12, 34, 35, 14, 57};
        int sum;
        sum = total(mark,5);
        printf(“Total=%d”, sum);
        return 0;
}
```

Array & Pointer

int a=5; a 5
100

int *b; b
200

b=&a; 100
200

b => 100

*b => *(b) => *(100) => 5

b+1 = b + 1 X sizeof(int)
= b + 1 X 2
= 100 + 2 = 102

0	1	2	3	4	5	6	7
99	67	78	56	88	90	34	85
1000	1002	1004	1006	1008	1010	1012	1014

int a[8];

a[0] => 99

*(1000) => 99

*(1000 + 0) => 99

a[3] => 56

*(1006) => 56

*(1000 + 3) => *(1000 + 3 X 2) => 56

a => 1000

a is the address of the first element of the array => 1000

a[i] => *(a + i)

Array & Pointer

```
void modify(int data[], int size){
        int i;
        data[3] = 54;
        data[1] = data[1] + 10;
}
int main(void) {
        int mark[5]={12, 34, 35, 14, 57};
        modify(mark,5);
        for(i=0 ; i<5 ; i++){
            printf(“%d ”, mark[i]);
        }
        return 0;
}
```

data	1000	size	5

Data[3] => *(data+3) => *(1000 + 3 X 2) => *(1006)

	0	1	2	3	4
mark	12	**44**	35	**54**	57
	1000	1002	1004	1006	1008

	0	1	2	3	4
mark	12	34	35	14	57
	1000	1002	1004	1006	1008

```
12   44   35   54   57
```

TWO DIMENSIONAL ARRAYS

- It has two subscripts
  - one subscript denotes row
  - the other denotes column.
- Declaration
  - int a[4][3];
  - Int a[4][3]={

```
              {5,3,7},
              {6,8,2},
              {1,7,9},
              {3,5,6}
          };
```

		Column 0	 1	 2
Row	0	a(0,0) 1000	(0,1) 1002	(0,2) 1004
	1	(1,0) 1006	(1,1) 1008	(1,2) 1010
	2	(2,0) 1012	(2,1) 1014	(2,2) 1016
	3	(3,0) 1018	(3,1) 1020	(3,2) 1022

5	3	7
6	8	2
1	7	9
3	5	6

TWO DIMENSIONAL ARRAYS

Column

Row	0	1	2
0	a[0][0] 5	a[0][1] 3	a[0][2] 7
1	a[1][0] 6	a[1][1] 8	a[1][2] 2
2	a[2][0] 1	a[2][1] 7	a[2][2] 9
3	a[3][0] 3	a[3][1] 5	a[3][2] 6

A[0][0]
A[0][1]
A[0][2]

```
for(j=0 ; j<3 ; j++){
        printf(“%d”, a[0][j]);
}
```

A[1][0]
A[1][1]
A[1][2]

```
for(j=0 ; j<3 ; j++){
        printf(“%d”, a[1][j]);
}
```

A[2][0]
A[2][1]
A[2][2]

```
for(j=0 ; j<3 ; j++){
        printf(“%d”, a[2][j]);
}
```

A[3][0]
A[3][1]
A[3][2]

```
for(j=0 ; j<3 ; j++){
        printf(“%d”, a[3][j]);
}
```

```
for(i=0 ; i<4 ; i++){
    for(j=0 ; j<3 ; j++){
        printf(“%d”, a[i][j]);
    }
}
```

TWO DIMENSIONAL ARRAYS

```
int I,j;
for(i=0 ; i<4 ; i++){
     for(j=0 ; j<3 ; j++){
          printf(“%d”, a[i][j]);
     }
}
```

```
int I,j;
for(i=0 ; i<4 ; i++){
     for(j=0 ; j<3 ; j++){
          scanf(“%d”, &a[i][j]);
     }
}
```

Column

Row	0	1	2
0	a[0][0] 5	a[0][1] 3	a[0][2] 7
1	a[1][0] 6	a[1][1] 8	a[1][2] 2
2	a[2][0] 1	a[2][1] 7	a[2][2] 9
3	a[3][0] 3	a[3][1] 5	a[3][2] 6

```
int I,j, sum=0;
for(i=0 ; i<4 ; i++){
     for(j=0 ; j<3 ; j++){
          sum=sum+a[i][j];
     }
}
```

TWO DIMENSIONAL ARRAYS

		Column			
		0	1	2	3
Row	0	a[0][0] 5	a[0][1] 3	a[0][2] 7	a[0][3] 4
	1	a[1][0] 6	a[1][1] 8	a[1][2] 2	a[1][3] 8
	2	a[2][0] 1	a[2][1] 7	a[2][2] 9	a[2][3] 1
	3	a[3][0] 3	a[3][1] 5	a[3][2] 6	a[3][3] 2

Problems

- Write a program to find the sum of elements present in each individual row and column.
- Write a program to find the sum of elements present in each diagonal.
- Write a program to fill a square matrix with value zero on the diagonals, 1 on the upper right triangle, and -1 on the lower left triangle.
- Write a menu-driven program to read and display an m X n matrix. Also find the sum, transpose, and product of two m X n matrices.
- Write a program to read and display a 2 X 2 X 2 array.

Matrix Multiplication

A (3 X 4)

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)

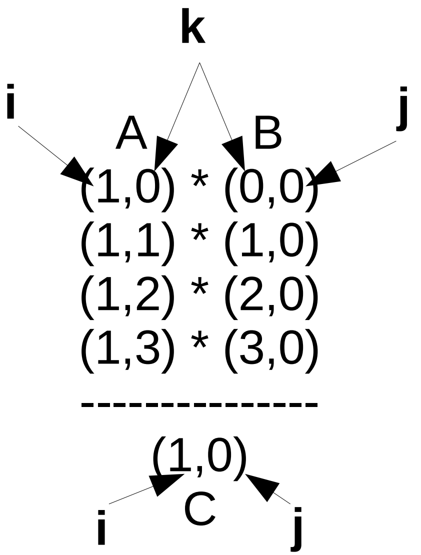
B (4 X 2)

(0,0)	(0,1)
(1,0)	(1,1)
(2,0)	(2,1)
(3,0)	(3,1)

```
sum=0;
for(k=0 ; k<4 ; k++){
    Sum = sum+A[i][k] * B[k][j];
}
C[i][j]=sum;
```

C (3 X 2)

(0,0)	(0,1)
(1,0)	(1,1)
(2,0)	(2,1)



(1,0) * (0,0)
(1,1) * (1,0)
(1,2) * (2,0)
(1,3) * (3,0)

(1,0)

```
for(i=0 ; i<3 ; i++){

    for(j=0 ; j<2 ; j++){


    }

}
```

```
int main(void){
        int i;
        for(i=0;i<5;i++){
                printf(“A ”);
        }
        return 0;
}
```

A A A A A

```
int main(void){
        int i;
        for(i=0;i<5;i++)
                printf(“A ”);
        return 0;
}
```

A A A A A

```
int main(void){
        int i;
        for(i=0;i<5;i++){
                printf(“A ”);
                printf(“B ”);
        }
        return 0;
}
```

A B A B A B A B A B

```
int main(void){
        int i;
        for(i=0;i<5;i++)
                printf(“A ”);
                printf(“B ”);

        return 0;
}
```

A A A A A B

String

- A string is a null-terminated character array

“abcd”

‘a’	‘b’	‘c’	‘d’	‘\0’

- Declaration
  - char <name>[size];
  - Example
    - char str[10];
    - char str[10]=”good”;

	str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]	str[9]
str	‘g’	‘o’	‘o’	‘d’	‘\0’					
	100	101	102	103	104	105	106	107	108	109

- Read String
  - scanf(“%s”,str);
- Write String
  - printf(“%s”,str);

String

```
int main(void)
{
	char str[10];
	scanf(“%s”,str);
	printf(“%s”,str);
	return 0;
}
```

```
int main(void)
{
	char str[10] = ”good”;
	printf(“%s”,str);
	return 0;
}
```

```
int main(void)
{
	char str[10];
	scanf(“%s”,str);
	for(i=0 ; str[i]!=’\0’ ; i++)
		;
	printf(“SIZE=%d”,i);
	return 0;
}
```

	str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]	str[9]
str	‘g’	‘o’	‘o’	‘d’	‘\0’					
	100	101	102	103	104	105	106	107	108	109

String length

String

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
a	‘g’	‘o’	‘o’	‘d’	‘\0’					

	b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]
b	‘d’	‘a’	‘y’	‘\0’						

	c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]
c	‘g’	‘o’	‘o’	‘d’	‘d’	‘a’	‘y’	‘\0’		

```
int main(void)
{
        char a[10], b[10], c[10];
        int i, k=0;
        scanf(“%s”,a);
        scanf(“%s”,b);
        for(i=0 ; a[i]!=’\0’ ; i++){
                c[k]=a[i];
                k++;
        }
        for(i=0 ; b[i]!=’\0’ ; i++){
                c[k]=b[i];
                k++;
        }
        c[k]=’\0’;

        printf(“%s”,c);
        return 0;
}
```

Concatenate two Strings

Problems on String

- **NOTE:**
  - **to receive a string without space**
    - **scanf(“%s”,a);**
  - **to receive a string without space**
    - **gets(a);**
  - **You can also directly initialize string during declaration**
    - **char a[100]=”asdf gh jkl vbcz”;**
- Write a program to find the length of a string.
- Write a program to convert characters of a string to upper case. (Do the reverse also)
- Write a program to concatenate two Strings.
- Write a program to reverse the given string.
- Write a program to compare two strings.
- Write a program to find whether a given string is a palindrome or not.
- Write a program to extract certain range of characters of a string. (Range is from one position to another position)
- Write a program to insert a string at the particular position of the the main text.
- Write a program to display position of the substring in the main string.
- Wriote a program to remove all the extra space in a text.
- Write a program to delete a substring from a text.
- Write program to display all the words present in the string.
- Write a program to remove all the continuous duplicate letters present in the string.
- Write a program to replace a pattern with another pattern in the text. (similar to find and replace)
- Write a program to enter a string. Then enter a new string. Then find whether the new string is the sub-string of the original string.

Structure

- Read the student information such as name, roll no, mark in subject-1, mark in subject-2, semester.

```
int main(void)
{
      char name[10];
      int rollno;
      float mark1, mark2;
      int sem;
      scanf(“%s”,name);
      scanf(“%d”,&rollno);
      scanf(“%f%f”,&mark1,&mark2);
      scanf(“%d”,&sem);

      printf(“%s : %d : %d : %f : %f” ,
  name,rollno,sem,mark1,mark2);
      return 0;
}
```

Structure

Data type

```
struct stu
{
        char name[10];
        int rollno;
        float mark1, mark2;
        int sem;
};
```

Member variable

User Defined data type
whose name is “**struct stu**”

```
scanf(“%s”,s1.name);
scanf(“%d”,&s1.rollno);
scanf(“%d”,&s1.mark1);

printf(“%d”,s1.rollno);
printff(“%s”,s1.name);
```

```
struct stu s1, s2;
```

Variable of
struct stu

```
s1.name
s1.rollno
.
.
S2.name
```

Structure is basically a user defined data type that can store related information (even of different data types) together.

Structure

```
struct stu
{
        char name[10];
        int rollno;
        float mark1, mark2;
        int sem;
};
```

```
struct stu s1={“ABC”,190,23.4,64.5,2};
```

- Write a program to enter two points and then calculate the distance between them.
- Write a program to read, display, add, and subtract two time defined using hour, minutes, and values of seconds.
- Write a program to read and display the information of all the students in the class. Then edit the details of the ith student and redisplay the entire information.
- Write a program to read and display information of a student, using a structure within a structure.

STRUCTURE

```
struct stu
{
        char name[10];
        int rollno;
        float mark1;
        int dob_dd;
        int dob_mm;
        int dob_yy;
};
```

```
struct stu s1,s2;
```

```
struct stu
{
        char name[10];
        int rollno;
        float mark1;
        int dob_dd;
        int dob_mm;
        int dob_yy;
} s1, s2;
```

NESTED STRUCTURES

```
struct stu
{
        char name[10];
        int rollno;
        float mark1;
        int dob_dd;
        int dob_mm;
        int dob_yy;
};
```

```
struct stu s1;

s1.name
s1.rollno
s1.mark1
s1.dob_dd
s1.dob_mm
s1.dob_yy
```

```
struct stu
{
        char name[10];
        int rollno;
        float mark1;
        struct dt{
                int dob_dd;
                int dob_mm;
                int dob_yy;
        }db;
};
```

```
struct stu s1;

s1.name
s1.rollno
s1.mark1
s1.db.dob_dd
s1.db.dob_mm
s1.db.dob_yy
```

NESTED STRUCTURES

```
struct stu
{
        char name[10];
        int rollno;
        float mark1;
        struct dt{
                int dob_dd;
                int dob_mm;
                int dob_yy;
        }db;
};

struct teacher
{
        char name[10];
        int emp_id;
        struct dt{
                int dob_dd;
                int dob_mm;
                int dob_yy;
        }db;
};
```

```
struct dt{
        int dob_dd;
        int dob_mm;
        int dob_yy;
};
struct stu
{
        char name[10];
        int rollno;
        float mark1;
        struct dt db;
};

struct teacher
{
        char name[10];
        int emp_id;
        struct dt db;
};
```

NESTED STRUCTURES

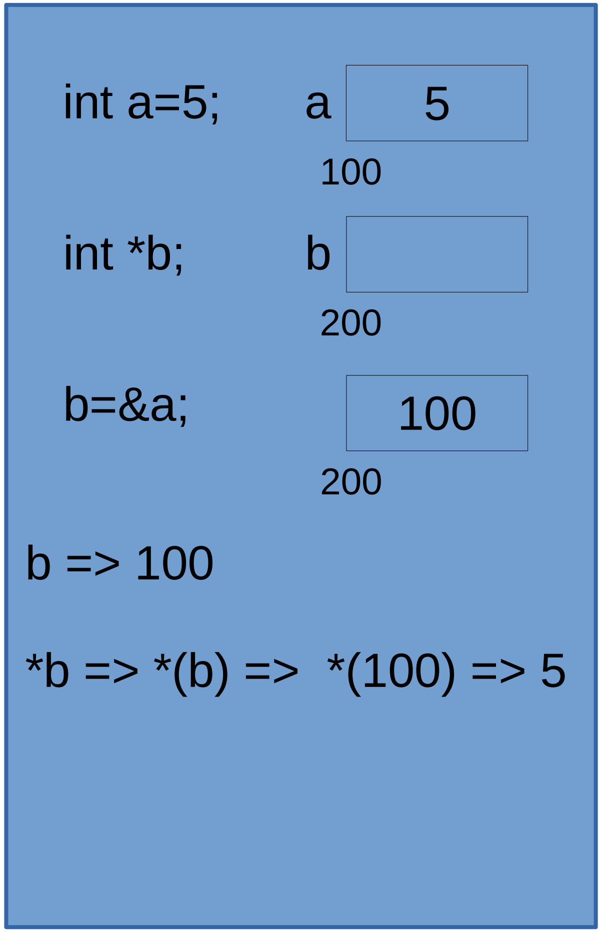
```
struct stu
{
        char name[10];
        int rollno;
        float mark1;
        struct dt{
                int dob_dd;
                int dob_mm;
                int dob_yy;
        }db;
};
```

```
struct stu s1;

s1.name
s1.rollno
s1.mark1
s1.db.dob_dd
s1.db.dob_mm
s1.db.dob_yy
```

```
struct dt{
        int dob_dd;
        int dob_mm;
        int dob_yy;
};
struct stu
{
        char name[10];
        int rollno;
        float mark1;
        struct dt db;
};
```

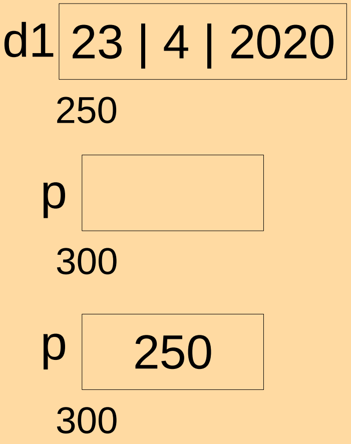
```
struct stu s1;

s1.name
s1.rollno
s1.mark1
s1.db.dob_dd
s1.db.dob_mm
s1.db.dob_yy
```

Structure & pointer



```
int a=5;        a [ 5 ]
                100

int *b;         b [   ]
                200

b=&a;             [ 100 ]
                200

b => 100

*b => *(b) =>  *(100) => 5
```

```
struct dt{
        int dob_dd;
        int dob_mm;
        int dob_yy;
};

struct dt d1={23,4,2020};

Struct dt *p;

p=&d1;
```



```
d1 [ 23 | 4 | 2020 ]
   250

p [     ]
  300

p [ 250 ]
  300
```

```
p->dob_dd   : 23
p->dob_mm : 4
p->dob_dd   : 2020

d1.dob_dd   : 23
d1.dob_mm : 4
d1.dob_dd   : 2020
```

Structure & Function

```
struct point{
	int x;
	int y;
};

void display(struct point p){
	printf(“%d %d”, p.x, p.y);
}
Int main(){
	struct point p1={23,5};
	display(p1);
	return 0;
}
```

```
struct point{
	int x;
	int y;
};

void display(struct point *p){
	printf(“%d %d”, p->x, p->y);
}
Int main(){
	struct point p1={23,5};
	display(&p1);
	return 0;
}
```

```
void display(int p){
	printf(“%d”, p);
}
Int main(){
	int p1=5;
	display(p1);
	return 0;
}
```

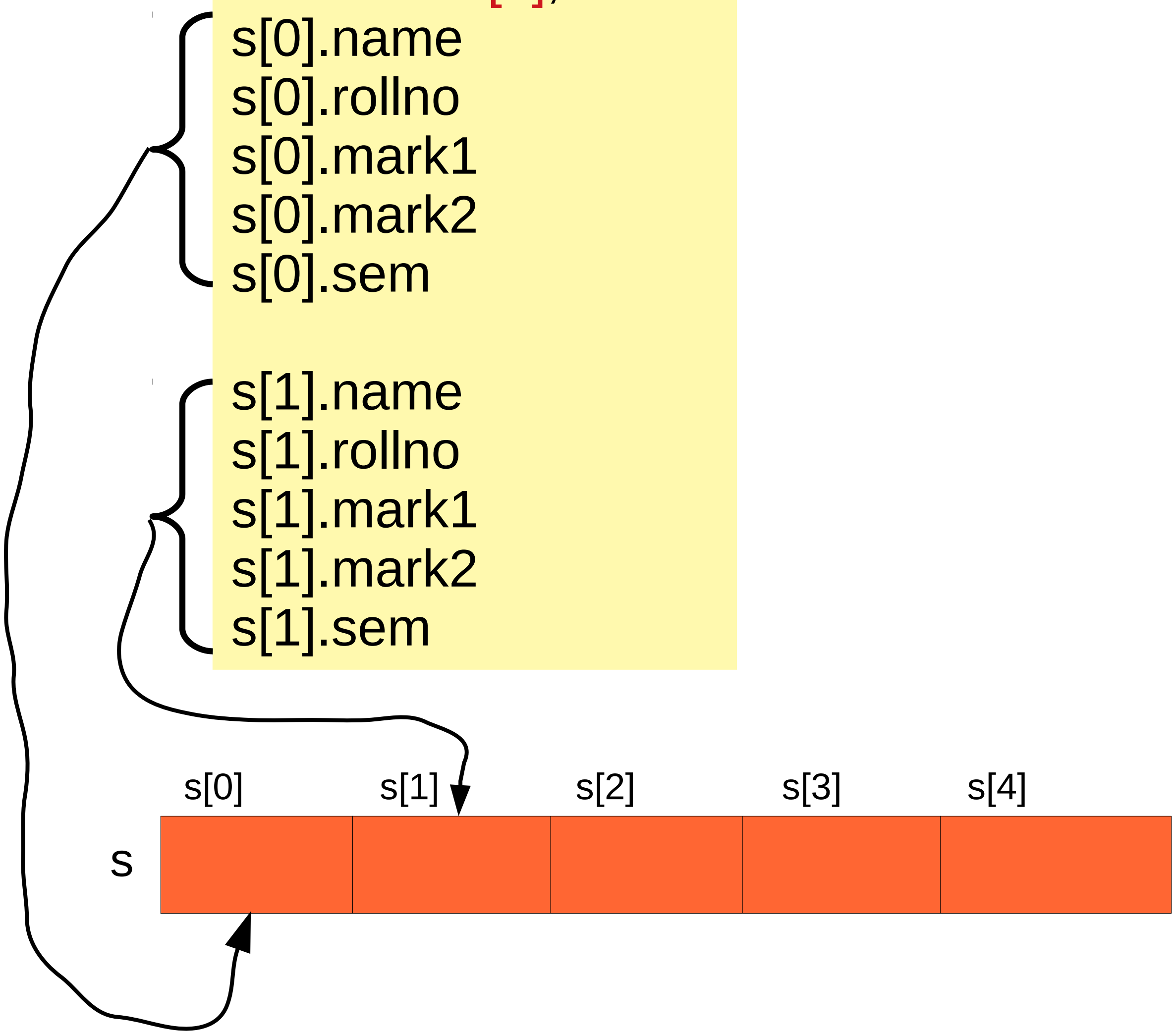
```
void display(int *p){
	printf(“%d”, *p);
}
Int main(){
	int p1=5;
	display(&p1);
	return 0;
}
```

Array of Structure

```
struct stu
{
        char name[10];
        int rollno;
        float mark1, mark2;
        int sem;
};
```

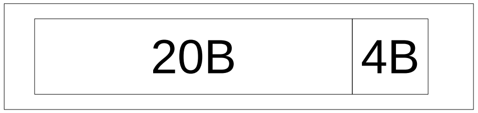
```
struct stu s;
s.name
s.rollno
smark1
s.mark2
s.sem
```

```
struct stu s[5];
s[0].name
s[0].rollno
s[0].mark1
s[0].mark2
s[0].sem

s[1].name
s[1].rollno
s[1].mark1
s[1].mark2
s[1].sem
```



Union

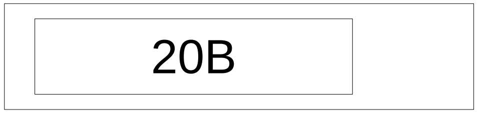
```
struct stu
{
	char name[20];
	int rollno;
};
```



20B 4B

```
struct stu s1;

s1.name
s1.rollno
```

```
union stu
{
	char name[20];
	int rollno;
};
```



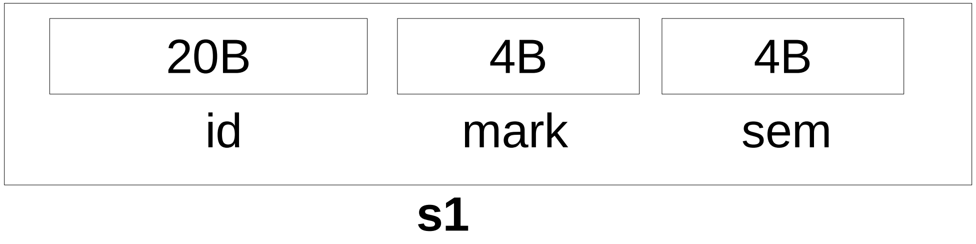
20B

a union is a collection of variables of different data types share a single block of memory

```
union stu s1;

s1.name
s1.rollno
```

Union

```
struct student
{
        union identification
        {
                char name[20];
                int rollno;
        }id;
        int mark;
        int sem;
};
```



```
struct student s1;

s1.id.name
s1.id.rollno
s1.mark
s1.sem
```

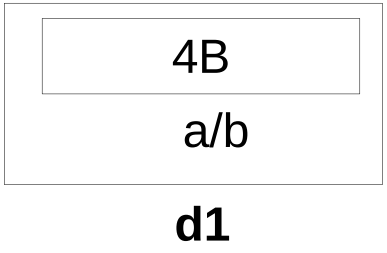
Only one will be used at a time

Union

```
union data
{
        char a;
        int b;
};
```

```
union data d1;
d1.b = 295;
printf(“%d”,d1.a);
```

→ 40



a/b	00101000	100
	00000001	101
	00000000	102
	00000000	103

296 <=> $\underbrace{00000000}\underbrace{00000000}\underbrace{00000001}\underbrace{00101000}$

enum Data Type

enum is used to declare named integer constants.

```
if(1){
      printf(“YES”);
}
```

```
enum ownconst {
FALSE,
TRUE
};
```

Not correct

```
if(TRUE){
      printf(“YES”);
}
```

```
if(TRUE){
      printf(“YES”);
}
```

```
enum tag_name {
enumeration_list
} variable_list;
```

enum Data Type

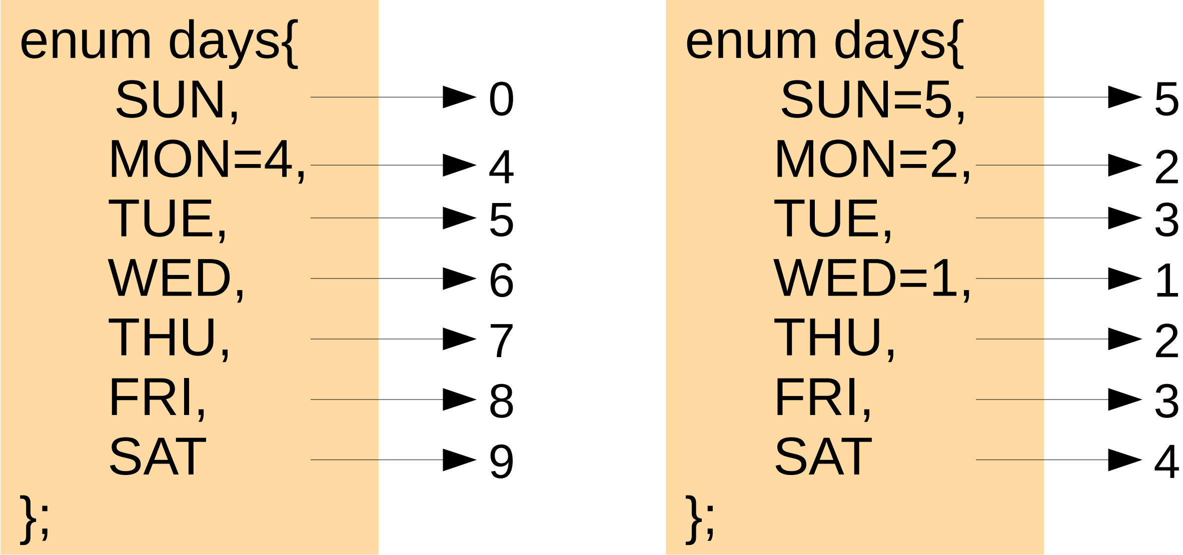
```
enum days{
      SUN,   → 0
      MON,   → 1
      TUE,   → 2
      WED,   → 3
      THU,   → 4
      FRI,   → 5
      SAT    → 6
};
```

```
enum days d1;
d1=MON;
d1=12;
```

```
int d2;
d2=TUE;
```

Size of enum data type is same as size of int data type

enum Data Type



```
enum days{
        SUN,
        MON=4,
        TUE,
        WED,
        THU,
        FRI,
        SAT
};
```

```
enum days{
        SUN=5,
        MON=2,
        TUE,
        WED=1,
        THU,
        FRI,
        SAT
};
```

Scope of the Variable

- Visibility of the variables
- Type
  - Block Scope
  - Function Scope
  - Program Scope
  - File Scope

```
void display(int p){
        int i=5;
        printf(“%d”, i);
}

Int main(){
        int k = 7;
        ~~printf(“%d”, i);~~
        printf(“%d”, k);
}
```

i is out side its scope

Block Scope

- A block refers to any sets of statements inside braces { and }.
- A variable declared inside a block has block scope.

```
int main(){
        int i=5;
        printf("%d\n",i);  ────► 5
        {
                int j=6;
                printf("%d\n",i);  ────► 5
                printf("%d\n",j);  ────► 6
        }
        printf("%d\n",i);  ────► 5
        ~~printf("%d",j);~~
        return 0;
}
```

```
int main(){
        int i=5;
        printf("%d\n",i);  ────► 5
        {
                int j=6,i=3;
                printf("%d\n",i);  ────► 3
                printf("%d\n",j);  ────► 6
        }
        printf("%d\n",i);  ────► 5
        return 0;
}
```

Function Scope

- It is active and visible from the beginning to the end of a function.
- Only the **goto label** has function scope.
- Goto satement and its corresponding label must be in the same function.

```
int main()
{
   int i;   /* block scope */
   .
   .
   mylb:    /* function scope */
   .
   .
   goto  mylb;
   .
   .
   return 0;
}
```

Program Scope

- It is declared outside a function and accessible through out the whole program.
- These variable are referred as global variable.

```
int x = 0;        /* program scope */

void count(){
      x++;
}

int main()
{
   x++;
   count();
   printf(“%d”,x);  → 2
   return 0;
}
```

```
int x = 2;        /* program scope */
float y = 4.5;    /* program scope */

void count(){
      printf(“%d, %f\n”,x,y);  → 2,  4.5
}

int main()
{
   int x=6;   /* block scope */
   count();
   printf(“%d, %f\n”,x,y);  → 6,  4.5
   {
      int y=8.5;   /* block scope */
      printf(“%d, %f\n”,x,y);  → 6,  8.5
   }
   return 0;
}
```

File Scope

- a global variable declared with the static specifier is said to have file scope.
- But a global variable without static can be accessed outside of a file.

b.c

```
int x=4;
static int y=7; /* file scope */
void dis(){
      printf("%d, %d\n",x,y);
}
```

a.c

```
int main(){
      extern int x;
      ~~extern int y;~~
      printf("%d, %d\n",x);        → 4
      ~~printf("%d, %d\n",y);~~
      return 0;
}
```

Storage Class

- It gives the scope (visibility) of variables and/or functions.
- It gives life time of variables and/or functions.
- It determines which part of the storage space will be allocated for that variable or function.
- It gives whether the variable will be automatically initialized or not.
- Types
  - auto specifier
  - static specifier
  - register specifier
  - extern specifier

auto specifier

- Accessible within the function or block in which it is declared
- Exists when the function or block in which it is declared is entered. Ceases to exist when the control returns from the function or the block in which it was declared
- By default the a variable with block scope is auto type.

```
void count(){
      auto int x = 1;
      x++;
      printf(“%d”,x); ────────► 2    2
}

int main()
{
   count();
   count();
   return 0;
}
```

static specifier

- Local
  - Accessible within the function or block in which it is declared
  - Retains value between function calls or block entries
  - automatically initialized

```
void count(){
      static int x = 1;
      x++;
      printf(“%d”,x);  ────────► 2    3
}

int main()
{
   count();
   count();
   return 0;
}
```

static specifier

- Global
  - Accessible within the file in which it is declared
  - Retains value
  - automatically initialized

```
static int x = 1;
void count(){
        x++;
        printf(“%d”,x);      → 3   4
}

int main()
{
   x++;
   count();
   count();
   return 0;
}
```

register Specifier

- Stores variables in registers (not guaranteed) may help to speed up your program.
- Accessible within the function or block in which it is declared
- Exists when the function or block in which it is declared is entered. Ceases to exist when the control returns from the function or the block in which it was declared
- It's illegal to take the address of a register variable.

```
int main()
{
   register int i;
   . . .
   for (i=0; i<MAX; i++){
      /* some statements */
   }
   . . .
   return 0;
}
```

extern Specifier

- The extern specifier provides a reference to a global variable defined elsewhere.
- The extern specifier declares outside or inside the function

b.c

```
int x=4;
void dis(){
        printf("%d \n",x);
}
```

a.c

```
extern int x;

int main(){
        printf("%d, %d\n",x);  ——————► 4
        return 0;
}
```

FEATURE	STORAGE CLASS			
	Auto	Extern	Register	Static
Accessibility	Accessible within the function or block in which it is declared	Accessible within all program files that are a part of the program	Accessible within the function or block in which it is declared	Local: Accessible within the function or block in which it is declared Global: Accessible within the program in which it is declared
Storage	Main Memory	Main Memory	CPU Register	Main Memory
Existence	Exists when the function or block in which it is declared is entered. Ceases to exist when the control returns from the function or the block in which it was declared	Exists throughout the execution of the program	Exists when the function or block in which it is declared is entered. Ceases to exist when the control returns from the function or the block in which it was declared	Local: Retains value between function calls or block entries Global: Preserves value in program files
Default value	Garbage	Zero	Garbage	Zero

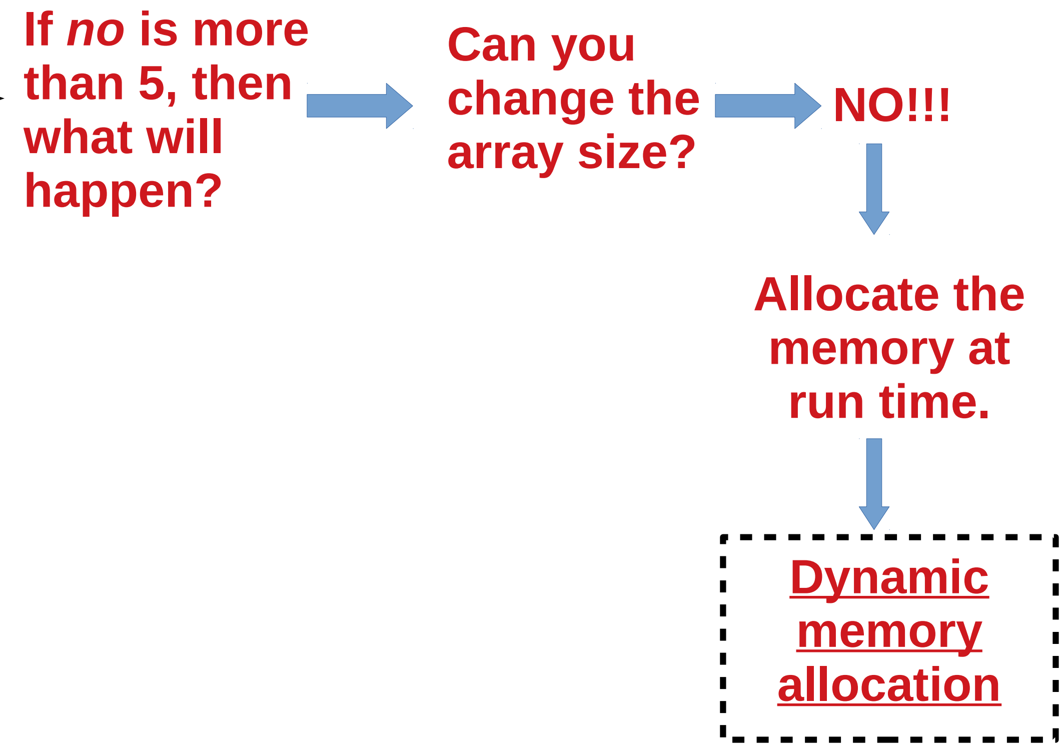
Constant Codifier

- A variable with the const modifier, the content of the variable cannot be changed after it is initialized.

```
int main()
{
	const int i=5;
	~~i=9;~~

	return 0;
}
```

Dynamic Memory Allocation

```
int main()
{
        int data[5];
        int no;
        printf("Enter no of elements:");
        scanf("%d",&no);
        for(i=0;i<no;i++)
                scanf("%d",&data[i]);
        return 0;
}
```



If *no* is more than 5, then what will happen?

Can you change the array size?

NO!!!

Allocate the memory at run time.

Dynamic memory allocation

Dynamic Memory Allocation

- How to allocate memory during run time?
  - malloc(),calloc(),realloc(),andfree()

p[i] = *(p+i)

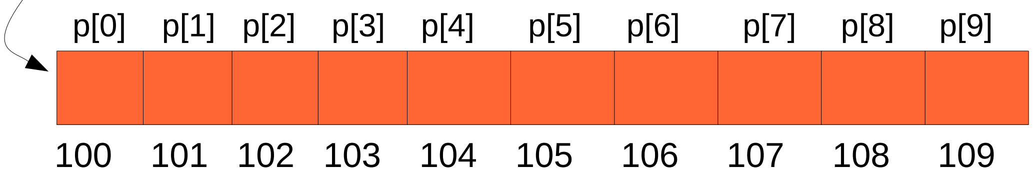
```
#include <stdlib.h>
void *malloc(size_t size);
```

Returns starting address of the allocated memory

How many Bytes of memory to be allocated

```
char *p;
p=malloc(10);
```

You can feel that p is a character array having 10 elements.



p[0]	p[1]	p[2]	p[3]	p[4]	p[5]	p[6]	p[7]	p[8]	p[9]
100	101	102	103	104	105	106	107	108	109

```
char *p;
int no;
printf("Enter no of elements:");
scanf("%d",&no);
p=malloc(no);
```

Dynamic Memory Allocation

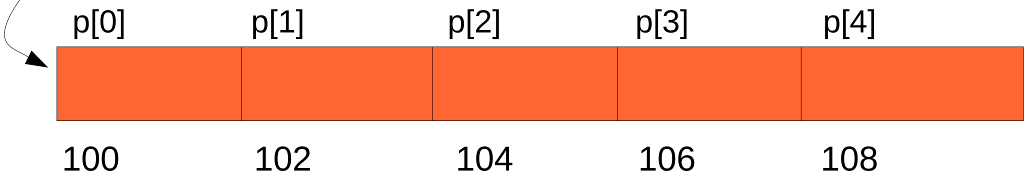
```
int *p;

int no;
printf(“Enter no of elements:”);
scanf(“%d”,&no);

p=malloc(no * sizeof(int));
```

5

```
int *p;

int no,i;
printf(“Enter no of elements:”);
scanf(“%d”,&no);

p=malloc(no * sizeof(int));
for(i=0;i<no;i++){
        p[i]=i+2;
}
```

```
int *p;

p=malloc(10);
```

sizeof(int)
=2 bytes



p[0] p[1] p[2] p[3] p[4]

100 102 104 106 108

p+1 = p+ 1 * sizeof(int)

```
int *p;

p=malloc(10);
free(p);
```

This will free the allocated memory

Dynamic Memory Allocation

Use of **calloc()**

```
int *p;

int no,i;
printf(“Enter no of elements:”);
scanf(“%d”,&no);

p=malloc(no * sizeof(int));
for(i=0;i<no;i++){
      p[i]=0;
}
```

```
    #include <stdlib.h>;

void *calloc(size_t nmemb, size_t size);
```

Number of element

Size of each element

```
int *p;

int no,i;
printf(“Enter no of elements:”);
scanf(“%d”,&no);

p=calloc(no, sizeof(int));
```

Unlike malloc, in calloc the memory is automatically set to zero.

Dynamic Memory Allocation

Use of **realloc()**

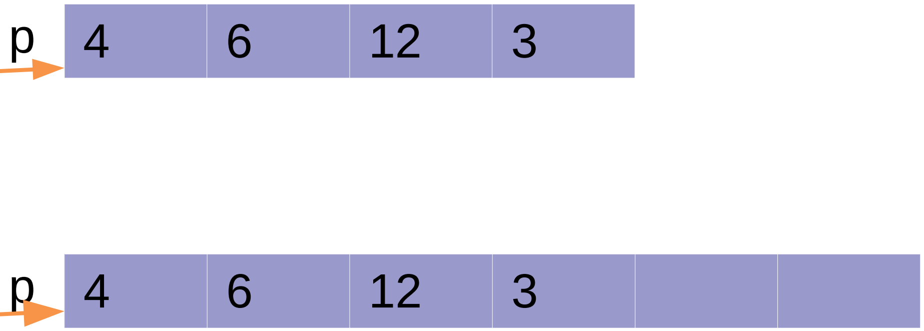
```
#include <stdlib.h>;

void *realloc(void *ptr, size_t size)
```

Pointer pointing to existing allocated memory

New memory size

realloc() changes the size of the memory block pointed to by ptr to size bytes.

```
char *p;

int no,i;
printf("Enter no of elements:");
scanf("%d",&no);

p=malloc(no);
.  .  .
printf("Enter new no of elements:");
scanf("%d",&no);
p=realloc(p, no);
```



File Handling

- File is a collection of data mostly stored in secondary storage.
- Basic operations on File
  - Open
  - Close
  - Read
  - Write
- File types
  - Text
    - contains ASCII characters only
  - Binary
    - can contain non-ASCII characters
    - Ex: image file, executable file, etc.

Open and Close a File

- Before performing any operations on a file, we need to open it.
  - fopen()
- After performing operations on a file, we need to close it.
  - fclose()
- Modes of opening a file

r	Open a text file for reading	**rb**	Open a binary file for reading
w	Create a text file for writing, if it exists, it is overwritten.	**wb**	Create a binary file for writing, if it exists, it is overwritten.
a	Open a text file and append text to the end of the file.	**ab**	Open a binary file and append data to the end of the file.

Open and Close a File

```
#include <stdio.h>
int main (){
   FILE *fp;
   fp = fopen("a1.txt", "r");
   if (fp == NULL) {
         printf("Error in file opening!\n");
         exit(0);
   }
   fclose(fp);
   return 0;
}
```

Name of the file

Opening mode

Read a File

```
#include <stdio.h>
int main (){
    FILE *fp;
    char c;
    fp = fopen("a1.txt", "r");
    if (fp == NULL) {
            printf("Error in file opening!\n");
            exit(0);
    }
    c=fgetc(fp);
    fclose(fp);
    return 0;
}
```

Read one character through **fp** pointer

```
#include <stdio.h>
int main (){
    FILE *fp;
    char c;
    fp = fopen("a1.txt", "r");
    if (fp == NULL)  exit(0);
    while(1){
            c=fgetc(fp);
            if(c==EOF) break;
            printf(“%c”,c);
    }
    fclose(fp);
    return 0;
}
```

Write to a File

```
#include <stdio.h>
int main (){
    FILE *fp;
    char c;
    fp = fopen("a1.txt", "w");
    if (fp == NULL) {
            printf("Error in file opening!\n");
            exit(0);
    }
    fputc(‘J’,fp);
    fclose(fp);
    return 0;
}
```

Write one character through **fp** pointer

Copy a File

```
#include <stdio.h>
int main (){
    FILE *fp, *fp1;
    char c;
    fp = fopen("a1.txt", "r");
    fp1 = fopen("a2.txt", "w");
    if (fp == NULL || fp1 == NULL)    exit(0);
    while(1){
            c=fgetc(fp);
            if(c==EOF) break;
            fputc(c,fp1);
    }
    fclose(fp);
    fclose(fp1);
    return 0;
}
```

Copy a Image File

```
#include <stdio.h>
#include <stdlib.h>
int main (){
   FILE *fp, *fp1;
   char c[1];
   fp = fopen("kiit.jpeg", "rb");
   fp1 = fopen("test.jpeg", "wb");
   if (fp == NULL || fp1 == NULL)   exit(0);
   while(!feof(fp)){
         fread(c,1,1,fp);
         fwrite(c,1,1,fp1);
   }
   fclose(fp);
   fclose(fp1);
   return 0;
}
```