(1)

* memory size cannot be modified during execution ex: array.

Dynamic memory allocation: memory is allocate while executing the program, at run time. memory size can be modified during execution

Ex: Linked list

Difference between malloc() and calloc():

malloc()	calloc()
* It allocates only single block of requested memory.	It allocates multiple blocks of requested memory.
* int *ptr; ptr = malloc(20 * sizeof(int)); for the above, 20 * 4 bytes of memory only allocated in one block. total = 80 bytes	int *ptr; ptr = calloc(20, 20 * sizeof(int)); for the above, 20 blocks memory will be created and each contains 20 * 4 bytes of memory. total = 1600 bytes.

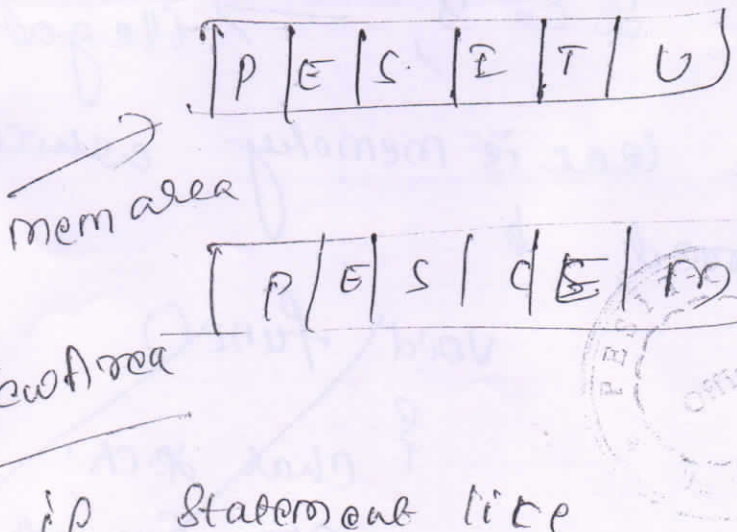
malloc()	calloc()
* malloc() doesn't initialize the allocated memory. It contains garbage values	calloc() initializes the allocated memory to zero.

Memory Leak

* memory overwrite * uninitialized memory * memory overread

Memory Leak

```
char *memoryArea = malloc(10);
char *newArea = malloc(10);
```



if statement like

memoryArea = newArea;

* In the code above, the developer has assigned the memoryArea pointer to the NewArea pointer. As a result the memory location to which memoryArea was pointing to becomes an orphan. It cannot be freed, as there is no reference to this location. This will result in a memory leak due to bytes.

Difference between a dangling pointer and memory leak

```
int *c = malloc(sizeof(int));
free(c);
*c = 3;  → illegal
```

A memory leak is memory which hasn't been freed, i.e.

```
void fun()
{ char *ch;
  ch = (char*) malloc(10);
}
```

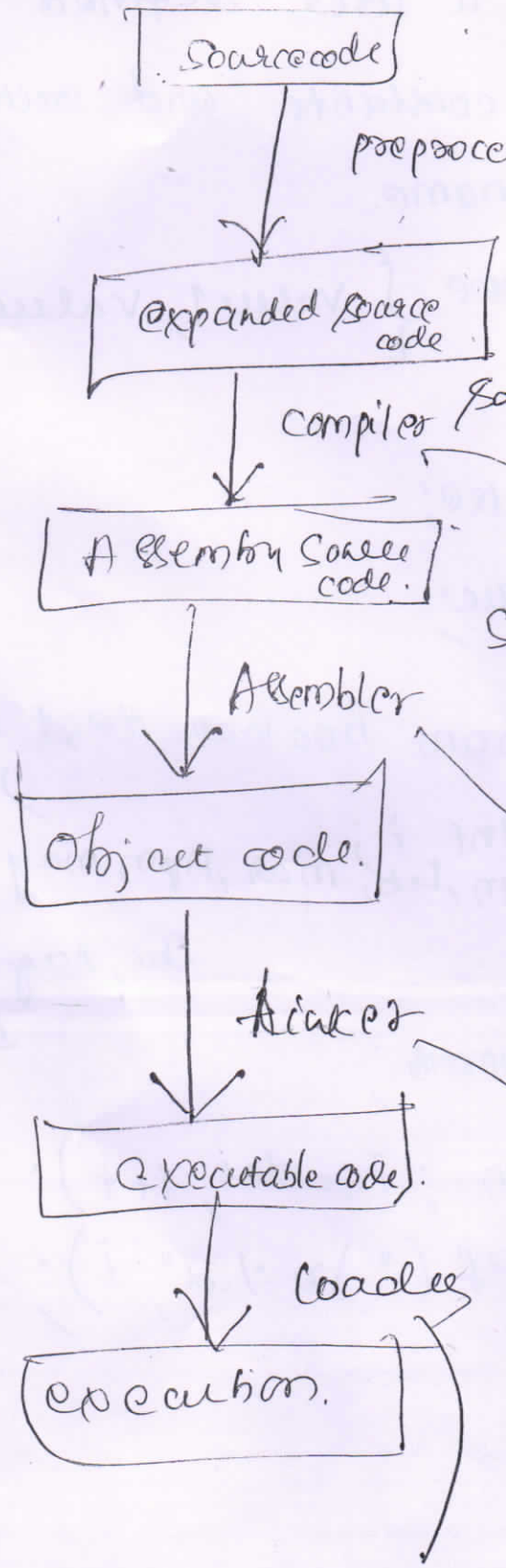
(7) (4)

Ex for dangling pointer:

```
char * c = malloc (16);
free(c);
c[1] = 'a';
```

* A memory leak is when you dynamically allocate memory from the heap but never free it, possibly because you lost all references to it

Source code: sample.c



preprocessor replaces
#define (macros)
#include (files),
conditional compilat
codes like #ifdef #ifnde
by their respective
values & source
codes in
source file.

compiler compiles
expanded source
code to assembly
source code.
it is a pgm that
converts assembly
source code
to obj code.

This is a program
that converts obj code
to executable code
& combines all object
codes together

Executable code is
loaded in CPU
& executed by loader.

C programming Enumeration.

An enumeration is a user defined datatype consists of integral constants and each integral constant is given a name.

enum type_name { value1, value2, ... valuen };

```
enum boolean {
        false;
        true;
};
        enum boolean test;
```

Ex:
```
void main()
{
    int i;
    enum month { Jan, Feb, Mar, Apr, May, Jun,
                 Jul, Aug, Sep, Oct,
                 Nov, Dec };
    enum month namamonth;
    for (i=Jan ; i<=dec; i++)
        Pf ("\n %d", i);
}
```

* depending on the language, the compiler could automatically assign default values to the enumerators - thereby hiding unnecessary detail from the programmer. These values may not even be visible to the programmer (information hiding).

```
# include <stdio.h>
```

(1) (8)

Structure padding: In order to align the data in memory, one or more empty bytes (addresses) are inserted (or left empty) between memory addresses which are allocated for other structure members while memory allocation. This is called structure padding

* Architecture of a computer processor is such that, it can read 1 word (4 bytes in 32 bit processor) from memory at a time.

* To make use of this advantage of processor, data are always aligned as 4 bytes package which leads to insert empty addresses b/w other members addresses

To avoid use #pragma pack(1) directive

VC++ supports this feature

Unions: 'C' union is also like a structure.

* Each element in a union is called a member.
* Structure allocates storage space for all its members separately.
* whereas, union allocates one common storage space for all its members.
* we can access only one member of union at a time.
* Structure occupies higher memory space.
* union occupies lower memory space over structure.
* we can access all members of structure at a time
* we can access only one member of union at a time.

(1)

pointer to an Array of Structures

* pointer variable can also store the address of the structure variable.
* pointer to Array of structure stores the base address of the structure array.

```
Struct Student
{
    Char name[20];
    Char subI [20];
    char sub2 [20];
    int Result;
} stud[4] = {{"parag", "sc", "so", 20},
             {"chethan", "se", "so", 40},
             {"Jeet", "sc", "so", 60},
             {"preet", "sc", "so", 80}
};
```

```
void main()
{ struct student *ptr = stud;
  for (i=0; i<4; i++)
    {pf("\n name: %s", i++),
     pf("\n sub1: %s", ptr->name);
     pf("\n sub2: %s", ptr->sub1);
     if (stud[i].result == 80)
       pf("good");
     else
       pf("avg")
     ptr++;
    }
}
```

The keyword typedef provides a mechanism for creating synonyms for previously defined data types.

```
typedef struct card Card;

Ex:
 typedef struct {
        char *face;
        char *suit;
    } Card;
```

---

enum:

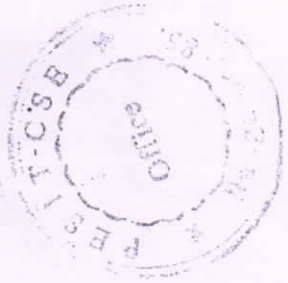
```
enum months { Jan, feb, mar,
              apr, may, Jun, July, Aug,
#include<stdio> Sep, oct, Nov, Dec };
  enum month{
      Jan=1, feb, mar ---);
int main()
  { enum months month;
char *monthname[] = { " ", "Jan", "feb",
  "mar", "apr", ... };
for (month = Jan; month <= Dec; month++)
  { pf("%d\n", month, monthname[month]);
    return 0 }.
```

(1)

* most C compilers will allow you to pass entire structures as parameters & return entire structure.

* As with all C parameters structures are passed by value and so if you want to allow a function to alter a parameter you have to remember to pass a pointer to a struct.

```
Struct complexadd (Struct comp a,
                   Struct comp b)
{
    Struct comp c;
    c.real = a.real + b.real;
    c.imag = a.imag + b.imag;
    Return c;
}
```



```
struct ~~start~~ stock
{ char name[20];
  int number;
};
struct stock fun();
int main()
{ struct ~~int~~ stock items;
  items = fun();
  pf("name in main\n");
  pf("\n%s\t", items.name);
  pf("%d\t", items.number);
  return 0;
}
struct stock fun()
{ struct stock items;
  pf("enter the item name\n");
  sf("%s", &items.name);
  pf("enter the no of items\n");
  sf("%d", &items.number);
  return items;
}
```

Sample program for calloc()

```
#include<stdio.h>
#include<stdlib.h>
int main()
{ int i,n;
int *a;
pf("no of elements to be entered");
sf("%d",&n);
a = (int *) calloc(n, sizeof(int));
pf("enter %d numbers:\n",n);
for(i=0; i<n; i++)
    { sf("%d",&a[i]);
    }
pf("the was entered are:");
for (i=0; i<n; i++)
    {pf("%d", a[i]);
    }
free(a);
return(0);
}
```

(3)

```
for (i=0; i<3; i++)
    { pf ("\n Roll no: %d", ph[i]->roll);
      pf ("\n Name: %s", ph[i]->name);
    }
  return(0);
}
```

Simple program on malloc).

```
#include <stdio.h>
#include <string.h>
int main()
{ char coords[] = "Here is a quick way to learn malloc";
  char *p;
  p = (char *) malloc (sizeof(coords));
  strcpy (p, coords);
  pf ("p = %s \n", p);
  pf ("coords = %s \n", coords);
  return 0;
}
```

(2)

```
#include <stdio.h>
    Struct Employee
    {   int id;
        Char name[25];
        int age;
        long salary;
    };
Employee Input ();
void main()
    { Struct Employee Emp;
      Emp = Input ();
      Pf ("\n employee id: %d", Emp.id);
      pf ("\n employee name %S" emp.name);
      Pf ("\n Employee Age : %d" emp.age);
      pf ("\n employee salary : %ld", emp.salary);
    }
Employee Input ()
      { Struct employee E;
        Pf ("\n enter employee id: ");
        Sf ("%d ", &E.id);
        Pf ("\n enter employee name");
        Sf ("%-S", &E.name);
        Pf ("\n Enter employee age : ");
        Sf ("%d", &E.age);
```

```
pf ("\n enter employee salary: ");
sf ("%ld", &e.salary);
return e;
}
```

Array of pointers to Array of structures.

```
#include <stdio.h>
struct stud
{ int roll;
  char name[10];
} *ptr[10];

int main()
{ int i;
  printf("\n Enter the Student details: ");
  for (i=0; i<3; i++)
  { ptr[i] = (struct stud *)
      malloc (sizeg (struct stud));
  pf ("\n enter the roll no: ");
  sf ("%d", &ptr[i]->roll);
  pf ("\n enter the name: ");
  sf ("%s", ptr[i]->name);
  } pf ("\n student details are: ");
```