PART A

1. What is the output if it is executed on a 32 bit processor?

```
void main()
{
   int *p;
   p = (int *)malloc(20);
   printf("%d", sizeof(p));
}
```

(a) 2
(b) 4
(c) 8
(d) Garbage value

Answer: (a) 2

2. When the pointer is NULL, then the function realloc is equivalent to the function _____

(a) alloc()
(b) calloc()
(c) free()
(d) malloc()

Answer: (d) malloc()

3. If malloc() and calloc() are not type casted, the default return type is _____

(a) void**
(b) void*
(c) int*
(d) char*

Answer: (b) void*

4. Which of the following functions allocates multiple blocks of memory, each block of the same size?

(a) malloc()
(b) realloc()

(c) calloc()

(d) free()

Answer: (c) calloc

5. What is the output?

```
Void main()
{
        Int *ptr;
        ptr-(int *)calloc(1,sizeof(int));
        *ptr=10;
        printf("%d\n",*ptr);
}
```

(a) 0

(b) -1

(c) 10

(d) NULL

Answer: (c) 10

6. Among 4 header files, which should be included to use the memory allocation functions like malloc(), calloc(), realloc() and free()?

(a) #include<string.h>

(b) #include<stdlib.h>

(c) #include<memory.h>

(d) Both b and c

Answer: (b) #include<stdlib.h>

7. Specify the 2 library functions to dynamically allocate memory?

(a) malloc() and memalloc()

(b) alloc() and memalloc()

(c) malloc() and calloc()

(d) memalloc() and faralloc()

Answer: (c) malloc() and calloc()

8. What is the Error of this program?

```
#include <stdio.h>

#include  <stdlib.h>

int main()

{

    char *ptr;

    *ptr = (char)malloc(30);

    strcpy(ptr, "RAM");

    printf("%s", ptr);

    free(ptr);

    return 0;

}
```

(a) Error: in strcpy() statement.
(b) Error: in *ptr = (char)malloc(30);
(c) Error: in free(ptr);
(d) No error
Answer: (b)Error: in *ptr = (char)malloc(30);

PART B

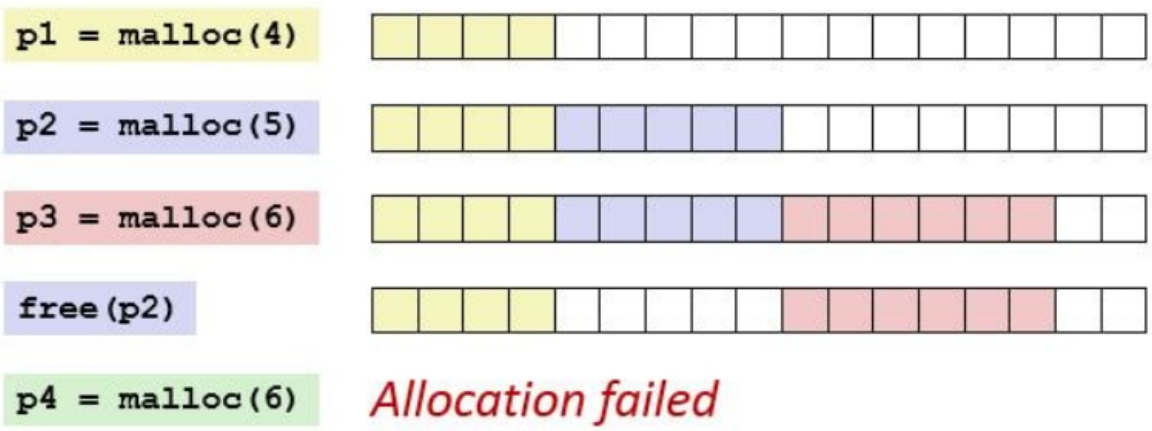
9. What is dynamic memory fragmentation?

The memory management function gives the guaranteed that allocated memory would be aligned with the object. The fundamental alignment is less than or equal to the largest alignment that's supported by the implementation without an alignment specification.

One of the major problems with dynamic memory allocation is fragmentation, basically, fragmentation occurred when the user does not use the memory efficiently. There are two types of fragmentation, external fragmentation, and internal fragmentation.

The external fragmentation is due to the small free blocks of memory (small memory hole) that are available on the free list but the program not able to use it. There are different types of free list allocation algorithms that used the free memory block efficiently.

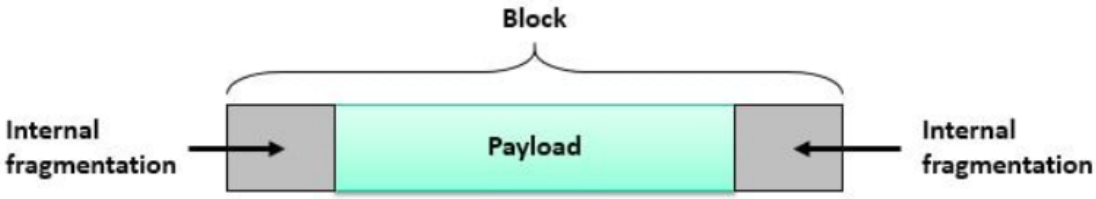
Consider a scenario where the program has 3 contiguous blocks of memory and the user frees the middle block of memory. In that scenario, you will not get a memory, if the required block is larger than a single block of memory. See the below Image,



The internal fragmentation is the wastage of memory that is allocated for rounding up the allocated memory and in bookkeeping (infrastructure). The bookkeeping memory is used to keep the information of allocated memory.

Whenever we called malloc function then it reserves some extra bytes (depend on implementation and system) for bookkeeping. This extra byte is reserved for each call of malloc and becomes a cause of the internal fragmentation.

If size of the payload is smaller than the block size, then internal fragmentation occur.



10. Summarize the advantages of dynamic memory allocation.

1. Allocation of memory
   In static memory allocation, space is provided at compile time, resulting in much memory waste. But in dynamic memory allocation, memory is assigned at run time so that the memory can be used accordingly with zero wastage of memory.
2. Efficiency
   Dynamic memory allocation is more efficient than static memory and flexible.
3. Reusability of memory
   The previously used memory can be used again using the free () function. This function frees the memory allocated before to use in the future.
4. Memory Reallocation
   In static memory allocation, we cannot change the size of any data structure if memory is allocated. But in dynamic memory allocation, we can change the size at our convenience by using the realloc() function.
5. Cost-effective
   Dynamic memory allocation is cost-effective as no extra cost is required for memory because memory is allocated per the user's needs.

11. List and explain dynamic memory allocation functions.

malloc() - Allocates memory and returns a pointer to the first byte of allocated space

calloc() - Allocates space for an array of elements, initializes them to zero and returns a pointer to the memory

free() - Frees previously allocated memory

realloc() - Alters the size of previously allocated memory

12. **What is the purpose of realloc( )?**

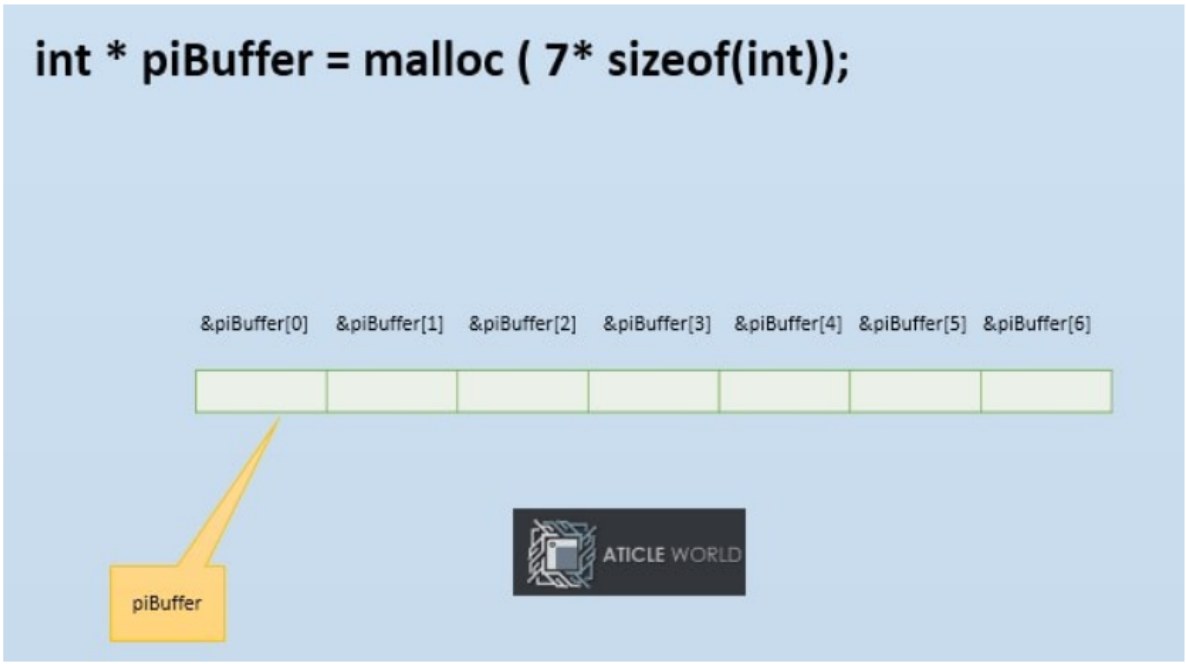
The realloc function is used to resize the allocated block of the memory. It takes two arguments first one is a pointer to previously allocated memory and the second one is the newly requested size. The realloc function first deallocates the old object and allocates again with the newly specified size. If the new size is lesser than the old size, the contents of the newly allocated memory will be the same as prior but if any bytes in the newly created object goes beyond the old size, the values of the object will be indeterminate.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main ()
{
   char *pcBuffer = NULL;
   /* Initial memory allocation */
   pcBuffer = malloc(8);
   strcpy(pcBuffer, "aticle");
   printf("pcBuffer = %s\n", pcBuffer);
   /* Reallocating memory */
   pcBuffer = realloc(pcBuffer, 15);
   strcat(pcBuffer, "world");
   printf("pcBuffer = %s\n", pcBuffer);
```

```
    free(pcBuffer);
    return 0;
}
```

PART C

13. Create a pointer to an integer and assign it heap memory. When memory is successfully assigned to the pointer then we can use this pointer as a 1D array and using the square braces "[]" we can access the pointer as like the statically allocated array. Let use the below Image for better understanding and write the program in C.




```
#include <stdio.h>
#include <stdlib.h>
#define FAIL 1
#define TRUE 0
int main(int argc, char *argv[])
{
```

```
int *piBuffer = NULL; //pointer to integer
int nBlock = 0; //Variable store number of block
int iLoop = 0; //Variable for looping
printf("\nEnter the number of block = ");
scanf("%d",&nBlock); //Get input for number of block
piBuffer = (int *)malloc(nBlock * sizeof(int));
//Check memory validity
if(piBuffer == NULL)
{
   return FAIL;
}
//copy iLoop to each block of 1D Array
for (iLoop =0; iLoop < nBlock; iLoop++)
{
   piBuffer[iLoop] = iLoop;
}
//Print the copy data
for (iLoop =0; iLoop < nBlock; iLoop++)
{
   printf("\npcBuffer[%d] = %d\n", iLoop,piBuffer[iLoop]);
}
// free allocated memory
free(piBuffer);
return TRUE;
```

}

Output:

Enter the number of block = 7

pcBuffer[0] = 0

pcBuffer[0] = 1

pcBuffer[0] = 2

pcBuffer[0] = 3

pcBuffer[0] = 4

pcBuffer[0] = 5

pcBuffer[0] = 6

14. Write a program to compute the sum of even numbers and odd numbers in a set of elements using dynamic memory allocation functions,

```
#include<stdio.h>
#include<stdlib.h>
void main(){
  //Declaring variables, pointers//
  int i,n;
  int *p;
  int even=0,odd=0;
  //Declaring base address p using malloc//
  p=(int *)malloc(n*sizeof(int));
  //Reading number of elements//
  printf("Enter the number of elements : ");
  scanf("%d",&n);
```

```
  /*Printing O/p -
  We have to use if statement because we have to check if memory
  has been successfully allocated/reserved or not*/
  if (p==NULL){
    printf("Memory not available");
    exit(0);
  }
  //Storing elements into location using for loop//
  printf("The elements are :
");
  for(i=0;i<n;i++){
    scanf("%d",p+i);
  }
  for(i=0;i<n;i++){
    if(*(p+i)%2==0){
      even=even+*(p+i);
    }
    else{
      odd=odd+*(p+i);
    }
  }
  printf("The sum of even numbers is : %d",even);
  printf("The sum of odd numbers is : %d",odd);
}
```

Output:

Enter the number of elements : 4

The elements are :

35

24

46

12

The sum of even numbers is : 82

The sum of odd numbers is : 35