Dynamic Memory Allocation

1

- In static memory allocation, the allocated memory size is fixed at compile time.
- Dynamic memory allocation is the process of allocating memory at run time.
- There are four library functions in C through which dynamic memory allocation can be done.
- These are malloc(), calloc(), realloc(), and free().
- We need to include the header file <alloc.h> in TC and the header file <malloc.h> in VC to use these functions.

2

size_t

- The **size_t** type is defined in stdio.h, stddef.h, stdlib.h, and string.h.
- This type is used for defining the sizes of strings and memory blocks. It is defined as:
- **typedef unsigned int size_t;**
- and is used as shown in the following example:

```
#include <string.h>
void lfunc (char *string)
{
    size_t string_length;
    string_length = strlen(string);
}
```

3

malloc()

- Declaration: **void *malloc(size_t size);**
- Remarks:
- malloc allocates a contiguous block of size bytes from the memory.
- It allows a program to allocate memory explicitly as it's needed, and in the exact amounts needed.
- Return Value:
- On success, malloc returns a pointer to the newly allocated block of memory.
- It returns a pointer of type void. This means that we can assign it to any type of pointer. It takes the following form:
  - **ptr=(cast-type*)malloc(byte-size);**
- Where ptr is a pointer of type cast-type. The malloc returns a pointer of cast-type to an area of memory with size byte-size.

4

- For examle:
- **x=(int*)malloc(100*sizeof(int));**
- On successful execution of this statement, a memory space equivalent to “100 times the size of an int” bytes is reserved and the address of the first byte of the memory allocated is assigned to the pointer x of type int.
- On error (if not enough space exists for the new block), malloc returns NULL.
- If the argument size == 0, malloc returns NULL.

5

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include <process.h>

main()
{
   char *str;

   /* allocate memory for string */
   if ((str = (char *) malloc(10*sizeof(char))) == NULL)
   {
      printf("Not enough memory to allocate buffer\n");
      exit(0);  /* terminate program if out of memory */
   }
```

6

```
/* copy "Hello" into string */
   strcpy(str, "Hello");

   /* display string */
   printf("String is: %s\n", str);

   /* free memory */
   free(str);
}
```

```
#include <stdio.h>
#include <malloc.h>
#include <process.h>
main()
{
   int *ptr;
   int n,i;
   printf("\nEnter the number of elements:");
   scanf("%d",&n);
   if ((ptr = (int *) malloc(n*sizeof(int))) == NULL)
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);
   }
   printf("\nEnter elements:\n");
   for(i=0;i<n;i++)
   {
       printf("\na[%d]=",i);
       scanf("%d",(ptr+i));
   }
   printf("\nArray contains:\n");
   for(i=0;i<n;i++)
   {
       printf("\na[%d]=%d",i,*(ptr+i));
   }
   /* free memory */
   free(ptr);
}
```

calloc()

- Declaration:

  void *calloc(size_t nitems, size_t size);

- Remarks:
- calloc( ) allocates a contiguous block (nitems * size) bytes and sets all bytes to zero.
- Return Value:
- On success, returns a pointer to the newly allocated block.
- On failure (not enough space exists for the new block, or nitems or size is 0), returns NULL.

9

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

 main( )
 {
    char *str;

    /* allocate memory for string */
    str = (char *) calloc(10, sizeof(char));

    /* copy "Hello" into string */
    strcpy(str, "Hello");

    /* display string */
    printf("String is %s\n", str);

    /* free memory */
    free(str);
 }
```

10

realloc()

- Declaration:

  void *realloc(void *block, size_t size);
- Remarks:
- realloc adjusts the size of the allocated block to size, copying the contents to a new location if necessary.

  Return Value:
- On success, realloc returns the address of the reallocated block, which might be different from the address of the original block.
- On failure (if the block can't be reallocated, or if size == 0), the function returns NULL.

11

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
void main()
{
  char *str;
  /* allocate memory for string */
  str = (char *) malloc(10*sizeof(char));
  //str = (char *) calloc(10,sizeof(char));
  if(str==NULL)
  {
     printf("\nNot enough memory");
     exit(0);
  }
  printf("Initially, String is: %s and  Address is: %u\n", str, str);
  /* copy "Hello" into string */
  strcpy(str, "Hello");
  printf("String is: %s and  Address is: %u\n", str, str);
  str = (char *) realloc(str, 20);
  printf("After realloc, String is: %s and New address is %u\n", str, str);
  /* free memory */
  free(str);
}
```

12

OUTPUT in VC

```
String is: Hello and  Address is: 4391264
After realloc, String is: Hello and New address is 4391264
Press any key to continue
```

OUTPUT in TC

```
String is: Hello and  Address is: 1472
After realloc, String is: Hello and New address is 1486
Press any key to continue
```

free()

- Declaration:

  `void free(void *block);`

- Remarks:
- free deallocates a memory block allocated by a previous call to calloc, malloc, or realloc.
- Return Value: None