Dynamic Memory Allocation using malloc, calloc and new

DMA = Dy. m/m allocation
m/m = memory.
Dy. = Dynamic.
arr = array.

} short terms

Understanding DMA:

Q Can you decide array size during runtime?

A In C – No
C++ – yes.

They r called Variable Length Arrays (VLA)

Q What r VLA?

A VLA are arrays for wh. array size is not specified during ~~runt~~ compile time; instead size is specified during runtime when program is run.

Eg
```
int n;
cin >>n;
int arr[n];
```
↓
arr size decided during runtime
↓
They are VLA

```
int n=10;
int arr[n];
```
↓
arr size known at compile time

Note:

- VLA r not created dynamically, ~~means~~
- They r not recommended bcoz m/m is allocated in stack & stackoverflow may occur.

Q What is the correct way to create dy. allocated arrays?

A Using Dy. m/m allocation (DMA)

- DMA means you allocate m/m during runtime.

  eg int ar[10] // Not DMA

  Here m/m is allocated for 10 element by runtime.

- This is possible bcoz arr size is known by runtime.

- In DMA, you do not allocate m/m by runtime, bcoz ~~arr~~ arr size is not known by runtime

- In DMA, m/m is allocated during runtime, bcoz ~~m/m~~ arr size is known at runtime.

Q Why is DMA done?
or
why do you use DMA?

A Sometime you r not able to specify the size of array by runtime.

So, You need to specify the size during runtime. So in that case DMA is done.

∴ Size of an is specified during runtime & that m/m ~~att~~ is allocated " " .

Q Ways to perform DMA?

A Using malloc → C, C++
calloc → C, C++
new → C++ only.

new is available in C++ only.

Malloc & Calloc fun

malloc = m/m allocation

Calloc = contiguous m/m alloc.

- C & C++ provide these two fun for DMA

How to use malloc

1) Specify the size of m/m you need in Bytes.

eg You need 10 elements of type int
Size of int is 4 Bytes
So you need 40 bytes.

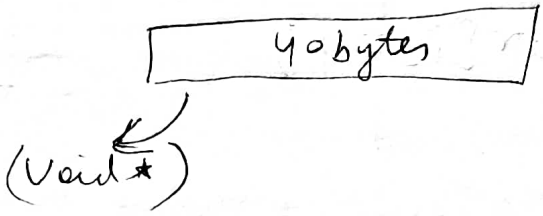
```
malloc(40);
```

2) This is prone to errors.
- Size of int may ~~change~~ vary from compiler to compiler
- So another way is to write like

```
malloc(10 * sizeof(int));
```

↓
This fun will calculate size of int data type.

3) Malloc will allocate, that much amount of m/m and return a void pointer (void *) pointing to start of that m/m block.



4) You need to convert (void *) to the pointer type required by you.
This is called type casting

```
(int *)malloc(10 * sizeof(int));
```

type casting

5) Assign the ptr returned by malloc to a ptr created by you.

```
int *ptr; // ptr created by u
ptr = (int *) malloc(10 * sizeof(int));
```

6) If u want to specify No. of el during runtime, then input the value of n from user and allocate that much m/m.

eg

```
int n;
cin >> n;        specify No. of element during runtime
int *ptr;
ptr = (int *) malloc(n * sizeof(int));
```

7) You can use this ptr, in the same way as an array.

```
for (int i=0; i<n; i++)
{
    ptr[i]=0;
}
```

Summary

- Steps to create dy. allocated arrays using malloc.

1) Create a pointer.

```
int * arr-ptr;
```

~~2) Allocate m/m using malloc~~

~~2) Input size from~~

2) Input No. of el in arr from user.

```
int n;
cin >> n;
```

3) Allocate m/m using malloc & assign to the ptr.

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

4) Use arr-ptr like an array.

```
arr-ptr[i] = i;
```

Calloc fun for DMA

- Using malloc is error prone as the value of ~~arr~~ arguments need to be correct.

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

- To overcome this, calloc is used.
- Calloc is used in similar manner as malloc. For convenience & to reduce mistakes, argument is split into two parts – n and size

Syntax:

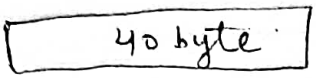
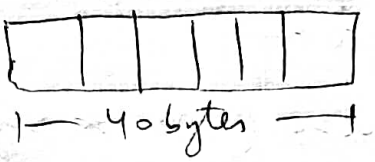
```
calloc(n, sizeof(int));
```

n ↓ : No. of el

sizeof(int) ↓ : size of each el.

Steps to use calloc:-

```
1) int * arr_ptr;          // create ptr
2) int n;                  // i/p no. of el.
   cin >> n;
3) arr_ptr = (int*) calloc(n, sizeof(int));
                           //use calloc.
4) for(int i=0; i<n; i++)
   {
       cout << arr_ptr[i]; // use arr_ptr like arr.
   }
```

malloc()	calloc()
1) Creates a single block of m/m 	1) Create multiple block for each element. 
2) Allocated m/m contains garbage.	2) allocated m/m is initialized to 0 by calloc.
3) No. of argument = 1	3) = 2
4) Faster, but not secure	4) Slower, but secure.
	Calloc is recommended over malloc

New operator for DMA

- For DMA, C++ provides an operator called new operator.
- It is more convenient than malloc and calloc.

Note :- new is an operator like +, -, = malloc, calloc - r fun.

How to use new operator

~~1) int * ptr;~~

1) int * arr_ptr; // Create a ptr.

2) int n;
   cin>>n; // I/P No. of el.

3) arr_ptr = new int [n];
   - new : type of array
   - [n] : No. of el.

   No need to use sizeof & type casting

4) for(int i=0; i<n; i++)
   {
   cout<< arr_ptr[i]
   }

Features of new operator

1) No need to use size of fun
New will determine the size of each element from the type you specified while ~~creating~~ using new operator.

2) No need to type cast the pointer returned by new
New will automatically type cast the ptr to the type you specified when using new operator.

Dynamic objects

- DO means u allocate m/m dynamically during runtime for obj.
- DO can be created using malloc, calloc or new.

Example to create an array of obj dynamically using new operator

Task: 1) To create an an aj obj.
- specify the no. aj el in an during runtime

2) assign, print, add values aj ~~obj~~ there dy. obj

```
#incln <iost->
using name-- std;
class student
{
    private:
    int rn, m1, m2;
    public:
    void set (int x, int y, int z);
    void get ();
    void add();
};
```

```
void stu::set(int x, int y, int z)
{
    rn = x;
    m1 = y;
    m2 = z;
}
void stu::get()
{
    cout << m1+m2 << endl;
}
void stu::add()
{
    cout << m1+m2;
}
int main()
{
    int n;
    cout << "How many elements do
    you want in an obj" ;
    cin >> n; // enter No. of el.
    //Create a ptr. of student type.
    Student *an_obj;
    //use new operator to allocate
    // m/m for n obj
    an_obj = new student[n];

    //This an_obj can be used like
      //array
```

```
for(int i=0; i<n; i++)
{
        int a, b, c;
        cout << "input values";
        cin >> a >> b >> c;
        an_obj [i].set (a, b, c);
}
  cout << "Print values";
for (int i=0; i<n; i++)
{
        an_obj [i].get ();
}

for (int i=0; i<n; i++)
{
        an_obj [i].add ();
}
```