Programming with C and C++

CSC-101 (Lecture 23)

Dr. R. Balasubramanian
Professor
Department of Computer Science and Engineering
Mehta Family School of Data Science and Artificial Intelligence
Indian Institute of Technology Roorkee
Roorkee 247 667

bala@cs.iitr.ac.in
https://faculty.iitr.ac.in/cs/bala/

Dynamic Array

- Dynamic arrays are a powerful data structure in programming that allows for creating and manipulating arrays of varying sizes during runtime.
- In C, the most commonly used memory allocation functions are **`malloc()`**, **`calloc()`**, and **`realloc()`**.

Dynamic Array

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int size = 5;
    int *arr = (int*) malloc(size * sizeof(int));
    int i;

        for(i = 0; i< size; i++) {
        arr[i] = i;
        }
    // Add a new element to the array
        size++;
        arr = (int*) realloc(arr, size * sizeof(int));
        arr[size-1] = i;

```

https://ideone.com/0bf596

Dynamic Array

```
16      for(i = 0; i< size; i++) {
17          printf("%d ", arr[i]);
18      }
19
20      free(arr);
21
22      return 0;
23  }
24
```

stdout

```
0 1 2 3 4 5
```

Dynamic Array

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int size = 5;
    int *arr = (int*) malloc(size * sizeof(int));
    int i;

        for(i = 0; i< size; i++) {
        arr[i] = i;
        }
    // Add a new element to the array
        size++;
        arr = (int*) realloc(arr, size * sizeof(int));
        arr[size-1] = i;

```

https://ideone.com/4U59Q7

Dynamic Array

```
16      for(i = 0; i< size; i++) {
17          printf("%d ", arr[i]);
18      }
19
20      free(arr);
21
22      printf("\n");
23
24      for(i = 0; i< size; i++) {
25          printf("%d ", arr[i]);
26      }
27
28      return 0;
29  }
30
```

stdout

```
0 1 2 3 4 5
0 0 -1289768944 21846 4 5
```

malloc (Memory Allocation)

- malloc stands for "memory allocation."
- It allocates uninitialized memory block.
- The content of the allocated memory is undefined and can contain garbage values.
- It takes one argument, which is the number of bytes to allocate.

calloc (Contiguous Allocation)

- calloc stands for "contiguous allocation."
- It allocates a block of memory for an array of elements, initializing all bits to zero.
- It ensures that the allocated memory is zero-initialized.
- It takes two arguments: the number of elements to allocate memory for and the size of each element in bytes.

calloc

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

int main() {
    int n = 10;  // Number of integers to allocate
    int *arr;

    // Allocate memory for n integers
    arr = (int *)calloc(n, sizeof(int));

    // Check if memory allocation was successful
    if (arr == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }
```

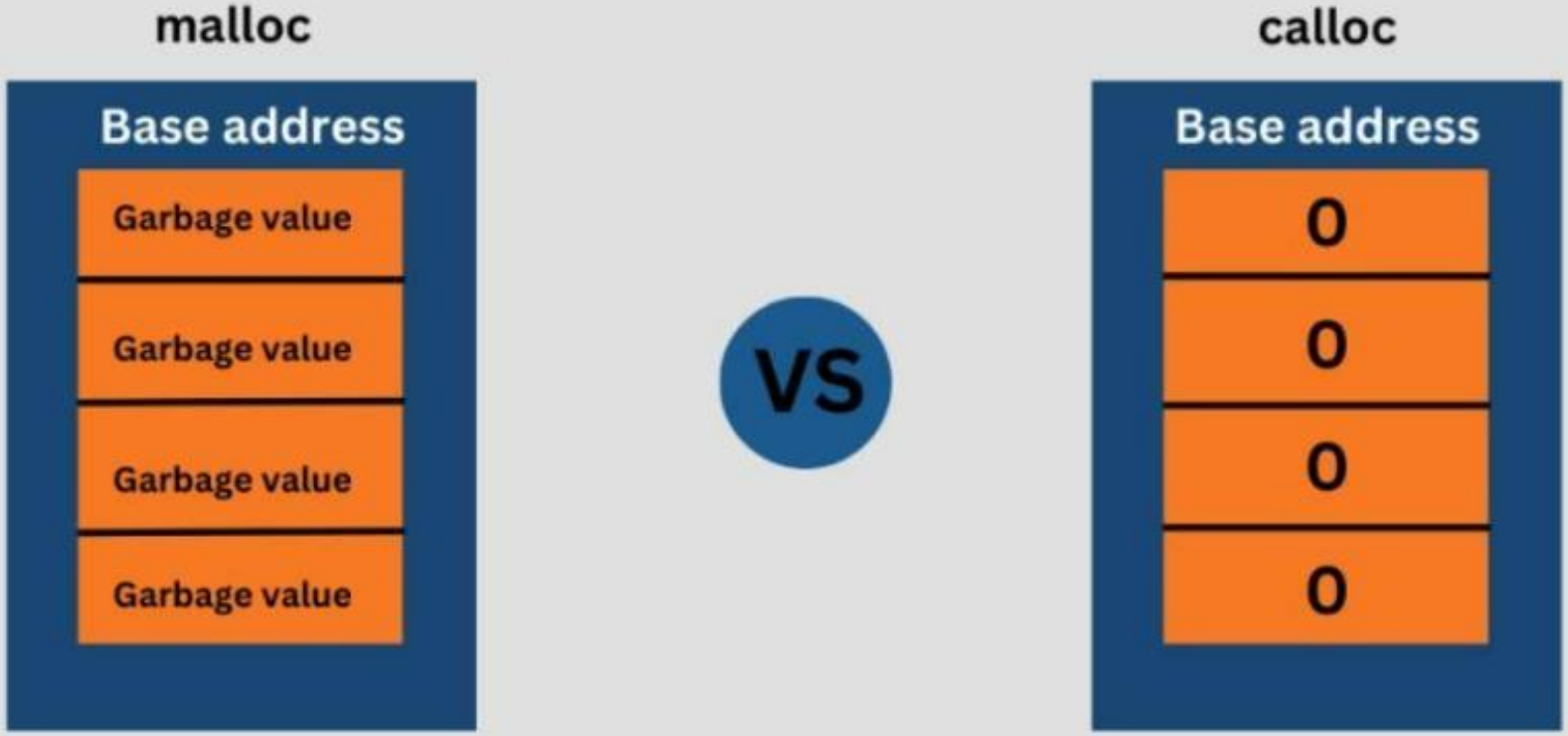
https://ideone.com/ATUIvD

```
16
17      // Use the allocated memory (for example, initialize the array)
18      for (int i = 0; i < n; i++) {
19          arr[i] = i * 10;
20          printf("%d ", arr[i]);
21      }
22
23      // Free the allocated memory (optional)
24      free(arr);
25
26      return 0;
27  }
28
```

stdout

```
0 10 20 30 40 50 60 70 80 90
```

Difference Between malloc() and calloc() with Examples



S.No.	malloc()	calloc()
1.	malloc() is a function that creates one block of memory of a fixed size.	calloc() is a function that assigns a specified number of blocks of memory to a single variable.
2.	malloc() only takes one argument	calloc() takes two arguments.
3.	malloc() is faster than calloc.	calloc() is slower than malloc()

S.No.	malloc()	calloc()
4.	Syntax : void* malloc(size_t size);	Syntax : Syntax : void* calloc(size_t num, size_t size);
5.	malloc() does not initialize the memory to zero	calloc() initializes the memory to zero

Pointer and reference variables

```
pv=0x7fffc9af9b88;
```

```
pv=0;
```

```
pv=NULL;
```

```
int v1=2*(*pv+v);
```

```
&w=2*(&u+&v);
```

```
int v2=2*(&u+&v);
```

Pointer and reference variables

```
pu+pv
```


```
*pu+*pv
```


```
pu=pu+1;
```


```
pv=*pu+pv;
```


```
pu=*pu+pv;
```


```
*pu=*pu+pu
```


void pointer in C

- int i=100; // integer variable initialization.
- int *ip; // integer pointer declaration.
- float *fp; // floating pointer declaration.
- void *ptr; // void pointer declaration.
- ip=fp; // incorrect.
- fp=&i; // incorrect

void pointer in C

- ptr=ip; // correct
- ptr=fp; // correct
- ptr=&i; // correct

Size of the pointers in C

https://ideone.com/L8OUlU

```c
#include <stdio.h>
int main()
{
    void *ptr = NULL; //void pointer
    int *p  = NULL;// integer pointer
    char *cp = NULL;//character pointer
    float *fp = NULL;//float pointer
    //size of void pointer
    printf("size of void pointer = %d\n\n",sizeof(ptr));
    //size of integer pointer
    printf("size of integer pointer = %d\n\n",sizeof(p));
    //size of character pointer
    printf("size of character pointer = %d\n\n",sizeof(cp));
    //size of float pointer
    printf("size of float pointer = %d\n\n",sizeof(fp));
    return 0;
}

```

stdout

```
size of void pointer = 8

size of integer pointer = 8

size of character pointer = 8

size of float pointer = 8
```

```c
#include <stdio.h>
#include<malloc.h>
int main()
{
    int m=1000;
    int *ptr = (int*)malloc(sizeof(int)) ;
    ptr=&m;
    printf("Value which is pointed by x pointer : %d",*ptr);

     return 0;
}

```

stdout

```
Value which is pointed by x pointer : 1000
```

Void pointer

```c
#include <stdio.h>

int main()
{
    int m=1000;
    void *ptr;
    ptr=&m;
    printf("Value which is pointed by x pointer : %d",*ptr);

     return 0;
}

```

compilation info

```
prog.c: In function ‘main’:
prog.c:8:54: warning: dereferencing ‘void *’ pointer
    printf("Value which is pointed by x pointer : %d",*ptr);
                                                     ^~~~
prog.c:8:54: error: invalid use of void expression
```

stdout

```
Standard output is empty
```

Type casting in pointer

https://ideone.com/y2dfJt

```c
#include <stdio.h>

int main()
{
    int m=1000;
    void *ptr;
    ptr=&m;
    printf("Value which is pointed by x pointer : %d",*(int*)ptr);

     return 0;
}

```

stdout

```
Value which is pointed by x pointer : 1000
```

```c
#include<stdio.h>
int main()
{
    float a[4]={6.1,2.3,7.8,9.0};
    void *ptr;
    ptr=a;
    ptr=ptr+12;
    printf("%f\n",*(float*)ptr);
}
```

stdout

```
9.000000
```

https://ideone.com/Kn90k4

Thank You!