BCSE102L- Structured and object-oriented programming

Module 3

Dr. P.Keerthika
Associate Professor
School of Computer Science & Engineering
VIT, Vellore.

BCSE102L- Structured and object-oriented programming

Module:1	**C Programming Fundamentals**	**2 hours**
Variables - Reserved words – Data Types – Operators – Operator Precedence - Expressions - Type Conversions - I/O statements - Branching and Looping: if, if-else, nested if, if-else ladder, switch statement, goto statement - Loops: for, while and do…while – break and continue statements.		
Module:2	**Arrays and Functions**	**4 hours**
Arrays: One Dimensional array - Two-Dimensional Array – Strings and its operations. User Defined Functions: Declaration – Definition – call by value and call by reference - Types of Functions - Recursive functions - Storage Classes - Scope, Visibility and Lifetime of Variables.		
Module:3	**Pointers**	**4 hours**
Declaration and Access of Pointer Variables, Pointer arithmetic – Dynamic memory allocation – Pointers and arrays - Pointers and functions.		
Module:4	**Structure and Union**	**2 hours**
Declaration, Initialization, Access of Structure Variables - Arrays of Structure - Arrays within Structure - Structure within Structures - Structures and Functions – Pointers to Structure -		
Module:5	**Overview of Object-Oriented Programming**	**5 hours**
Features of OOP - Classes and Objects - "this" pointer - Constructors and Destructors - Static Data Members, Static Member Functions and Objects - Inline Functions – Call by reference - Functions with default Arguments - Functions with Objects as Arguments - Friend Functions and Friend Classes.		
Module:6	**Inheritance**	**5 hours**
Inheritance - Types of Inheritance: Single inheritance, Multiple Inheritance, Multi-level Inheritance, Hierarchical Inheritance - Multipath Inheritance - Inheritance and constructors.		
Module:7	**Polymorphism**	**4 hours**
Function Overloading - Operator Overloading – Dynamic Polymorphism - Virtual Functions - Pure virtual Functions - Abstract Classes.		
Module:8	**Generic Programming**	**4 hours**
Function templates and class templates, Standard Template Library.		
	Total Lecture hours:	**30 hours**

BCSE102L- Structured and object-oriented programming – Text Books and Reference Books

Text Book(s)	
1.	Herbert Schildt, C: The Complete Reference, 4th Edition, McGraw Hill Education, 2017
2.	Herbert Schildt, C++: The Complete Reference, 4th Edition, McGraw Hill Education, 2017.
Reference Books	
1.	Yashavant Kanetkar, Let Us C: 17th Edition, BPB Publicaitons, 2020.
2.	Stanley Lippman and Josee Lajoie, C++ Primer, 5th Edition, Addison-Wesley publishers, 2012.

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

BCSE102P- Structured and object-oriented programming Laboratory

	Indicative Experiments
1.	Programs using basic control structures, branching and looping
2.	Experiment the use of 1-D, 2-D arrays and strings and Functions
3.	Demonstrate the application of pointers
4.	Experiment structures and unions
5.	Programs on basic Object-Oriented Programming constructs.
6.	Demonstrate various categories of inheritance
7.	Program to apply kinds of polymorphism.
8.	Develop generic templates and Standard Template Libraries.

Text Book(s)	
1.	Robert C. Seacord, Effective C: An Introduction to Professional C Programming, 1st Edition, No Starch Press, 2020.
Reference Book(s)	
1.	Vardan Grigoryan and Shunguang Wu, Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20's latest features, 1st Edition, Packt Publishing Limited, 2020.

BCSE102L- Structured and Object-Oriented Programming

- **Module-3: POINTERS**
  - **Declaration and Access of Pointer Variables**
  - **Pointer Arithmetic**
  - **Dynamic Memory Allocation**
  - **Pointers and Arrays**
  - **Pointers and Functions**

Dynamic Memory Allocation

- Since C is a structured language, it has some fixed rules for programming.
- Allocation of memory during execution /run time.
- **For example : How to change the size of an array in run time??.**

Elements	X[0]	X[1]	X[2]	X[3]	X[4]
Value	1	2	3	4	5
Address	1000	1002	1004	1006	1008

- **When limited storage only required??**
- **When Extra storage needed??**
- Leads to Dynamic Memory Allocation
- 4 library functions - defined under **<stdlib.h>** header file to facilitate dynamic memory allocation

Difference between Dynamic Memory Allocation and Static Memory Allocation

Static Memory Allocation	Dynamic Memory Allocation
Allocates the memory at load time	Allocates the memory at run time.
Memory is fixed, so increasing/decreasing is not possible at run time	Memory is flexible, so increasing/decreasing is possible at run time
Deallocated when life time is over.	Possible to deallocate the memory at runtime using "free" function.
Faster in Execution	Slower in execution

Memory Allocation

Function	Use of Function
Malloc()	Allocates requested size of bytes and returns a pointer first byte of allocated space
Calloc()	Allocates space for an array elements, initializes to zero and then returns a pointer to memory
Free()	Deallocate the previously allocated space
Realloc()	Change the size of previously allocated space

malloc() method

- Dynamically allocate a **single large block of memory** with the specified size.
- It returns a pointer of **type void** which can be cast into a pointer of any form.
- It doesn't Initialize memory at execution time so that it has initialized each block with the default garbage value initially.

SYNTAX:

ptr = (cast-type*) malloc(byte-size)

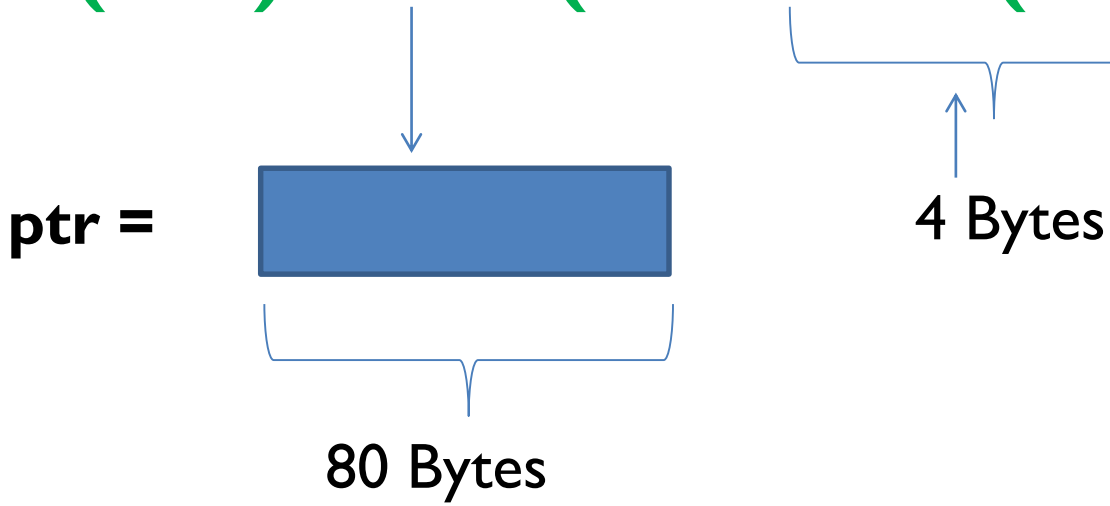
EXAMPLE:

ptr = (int*) malloc(5 * sizeof(int));

malloc() method

Example:

```
int *ptr = (int*)malloc(20 * sizeof(int));
```



80 bytes of memory block dynamically allocated to ptr

Calloc() method

- Dynamically allocate the specified number of blocks of memory of the specified type.
- Similar to malloc() but has two different points
  - It initializes each block with a default value '0'.
  - It has two parameters as compare to malloc().

SYNTAX:

ptr = (cast-type*) Calloc(N,Element size)

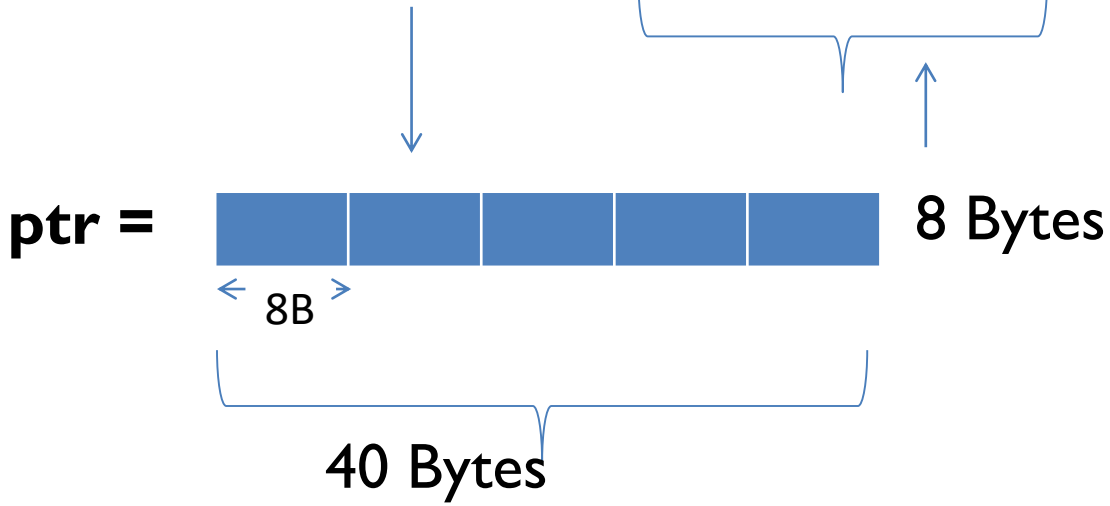
EXAMPLE:

ptr = (int*) calloc(5, sizeof(int));

calloc() method

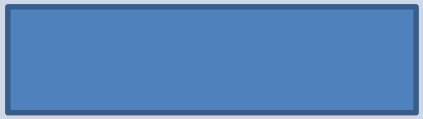
Example:

```
double *ptr = (double*)calloc(5, sizeof(double));
```



5 blocks of 8 bytes each is dynamically allocated to ptr

Difference between malloc and calloc

Malloc	Calloc
	
Allocates the memory bytes as a single block	Allocates the memory bytes as a multiple blocks
No Initialization – Produces Garbage Values	Initialized to 0 in all blocks
Faster in Execution	Slower in execution

free() method

- **Dy**namically **de-allocate** the memory.
- Memory allocated using functions malloc() and calloc() is not de-allocated on their own.
- free() method is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

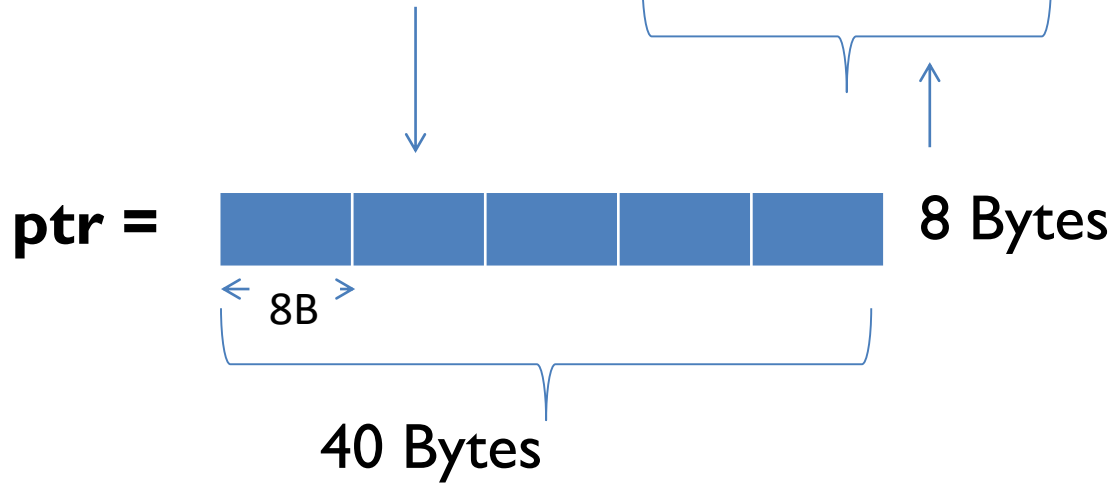
SYNTAX:

```
free(ptr);
```

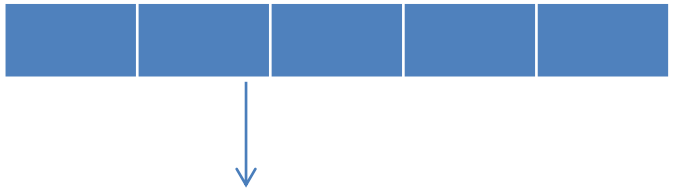
free() method

Example:

```
double *ptr = (double*)calloc(5, sizeof(double));
```



- 5 blocks of 8 bytes each is dynamically allocated to ptr
- **free(ptr);**



Memory of ptr is released

realloc() method

- Dynamically change the memory allocation of a previously allocated memory.
- In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to **dynamically re-allocate memory**.
- Re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value.

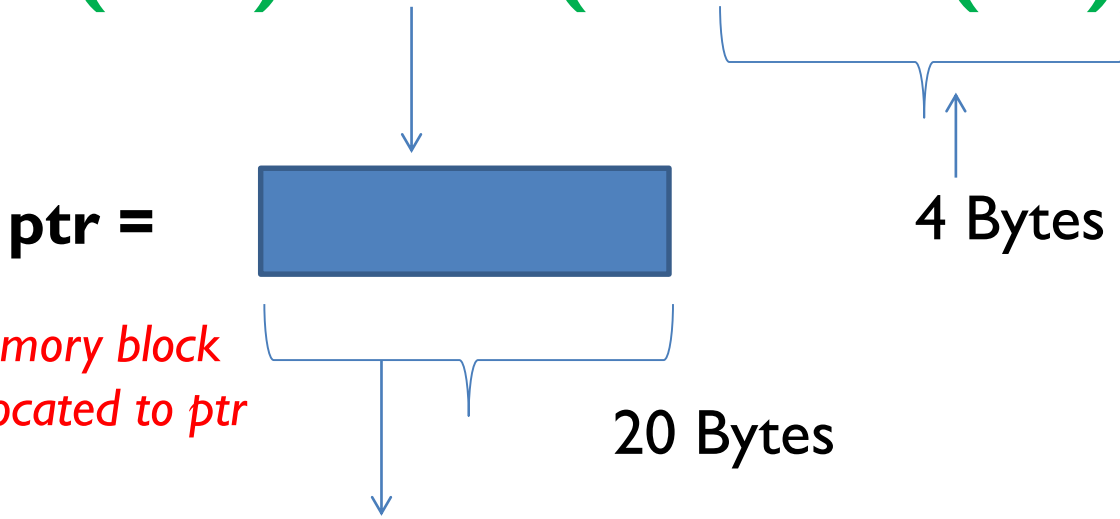
SYNTAX:

```
ptr = realloc(ptr, newSize);
```

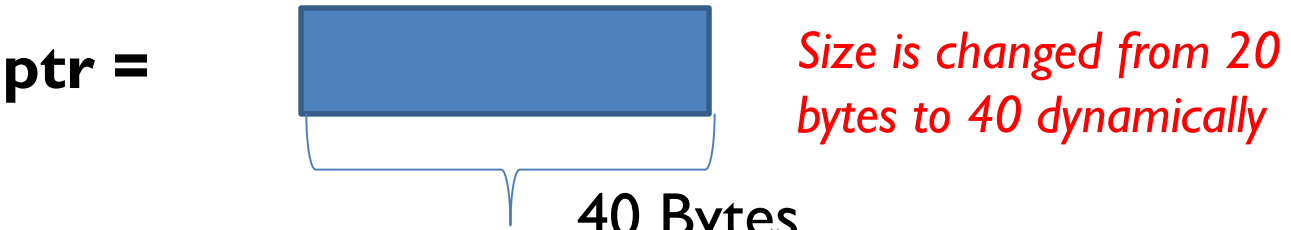
Realloc() method

Example:

```
int *ptr = (int*)malloc(5 * sizeof(int));
```




```
ptr = realloc(ptr, 10 * sizeof(int));
```



Example: Sum of n numbers

```
int main()
{
        int n, i, *p, sum = 0;
        printf("Enter number of elements: ");
        scanf("%d", &n);
        p = (int*) malloc(n * sizeof(int)); // (int*) calloc(n, sizeof(int));
        if(p == NULL)
        {
                printf("memory not allocated.");
                exit(0);
        }
        printf("Enter elements: ");
        for(i = 0; i < n; ++i)
        {
                scanf("%d", p+ i);
                sum = sum + *(p+ i);
        }
        printf("Sum = %d", sum);
        free(p);  return 0;
}
```

```
Enter number of elements: 5
Enter elements: 1
2
3
4
5
Sum = 15
```

Example: Realloc

```
int main()
{
        int *p,i , n1, n2;
        printf("Enter size of array: ");
        scanf("%d", &n1);
        p = (int*) malloc(n1 * sizeof(int));
        printf("Addresses of allocated memory: ");
        for(i = 0; i < n1; ++i)
                printf("%u\n",p+ i);
        printf("\nEnter new size of array: ");
        scanf("%d", &n2);
        p = realloc(p, n2 * sizeof(int));
        printf("Addresses of newly allocated memory: ");
        for(i = 0; i < n2; ++i)
                printf("%u\n", p+ i);
        return 0;
}
```

```
Enter size of array: 3
Addresses of allocated memory: 6976752
6976756
6976760

Enter new size of array: 5
Addresses of newly allocated memory: 6976752
6976756
6976760
6976764
6976768
```

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Thank You