Problem Solving With C - UE24CS151B

Dynamic Memory Management

Prof. Sindhu R Pai
PSWC Theory Anchor, Feb-May, 2025
Department of Computer Science and Engineering

PROBLEM SOLVING WITH C

Dynamic Memory Management

PES UNIVERSITY

1. Problem with the Arrays
2. Memory Allocation
3. Dynamic allocation
4. Use of malloc(), calloc(), realloc(), free()

PES UNIVERSITY

Problem with the Arrays

- Few situations while coding:
  - Amount of data cannot be predicted beforehand
  - Number of data items keeps changing during program execution

- In such cases, use of fixed size array might create problems:
  - Wastage of memory space (under utilization)
  - Insufficient memory space (over utilization)

- **Example:** A[1000] can be used but what if the user wants to run the code for only 50 elements
  //memory wasted

Solution: Can be avoided by using the concept of Dynamic memory management

PES UNIVERSITY

Memory Allocation

1. **Static allocation**
   - decided by the compiler
   - allocation at load time [before the execution or run time]
   - example: variable declaration (int a, float b, a[20];)

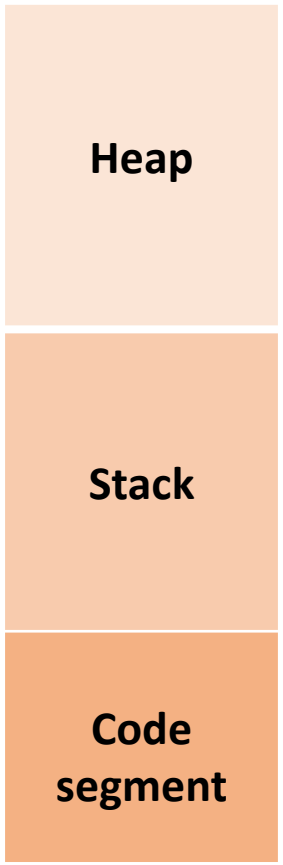
2. **Automatic allocation**
   - decided by the compiler
   - allocation at run time
   - allocation on entry to the block and deallocation on exit
   - example: function call (stack space is used and released as soon as callee function returns back to the calling function)

3. **Dynamic allocation**
   - code generated by the compiler
   - allocation and deallocation on call to memory allocation and deallocation functions

PES UNIVERSITY

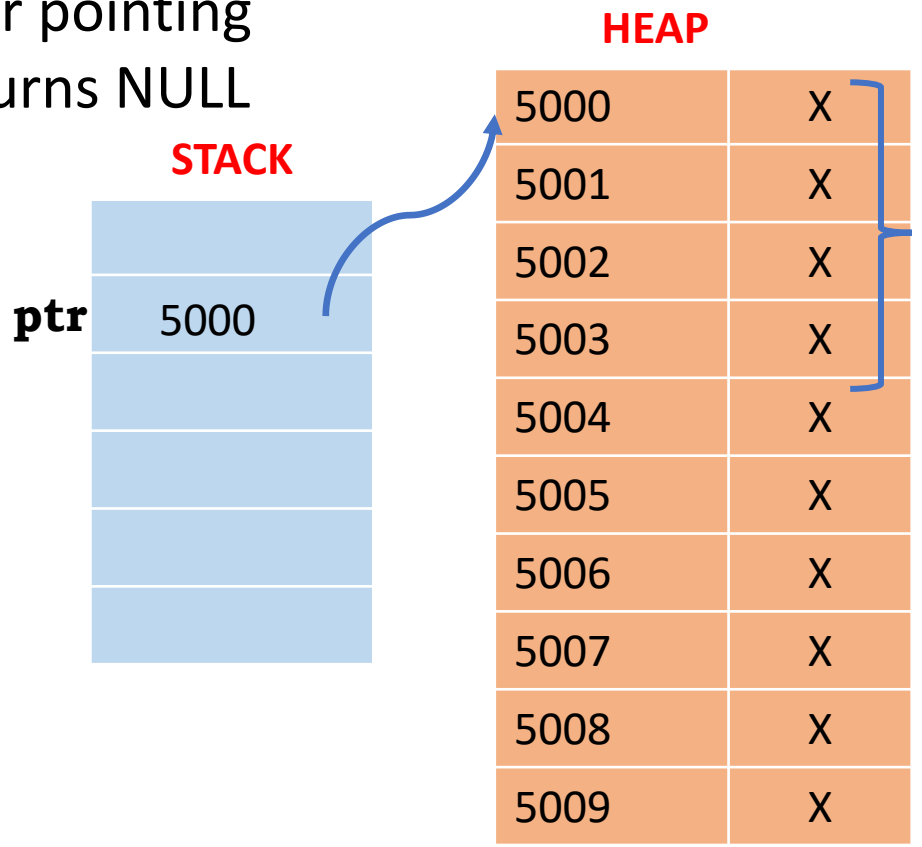
Dynamic Allocation

- Process of allocating memory at runtime/execution
- Uses the **Heap region of Memory segment**
- No operator in C to support dynamic memory management
- Library functions are used to dynamically allocate/release memory
  - malloc()
  - calloc()
  - realloc()
  - free()
- Available in stdlib.h



malloc() - memory allocation

- Allocates requested size of bytes and returns a void pointer pointing to the first byte of the allocated space on success. Else returns NULL
- The return pointer can be type-casted to any pointer type
- Memory is not initialized
- **Syntax**:
  void *malloc(size_t N); // Allocates N bytes of memory
- **Example:**
  int* ptr = (int*) malloc(sizeof (int)); // Allocate memory for an int
- Coding example



calloc() - contiguous allocation

- Allocates space for elements, initialize them to zero and then returns a void pointer to the memory. Else returns NULL
- The return pointer can be type-casted to any pointer type
- **Syntax**:

```
void *calloc(size_t nmemb, size_t size);
```

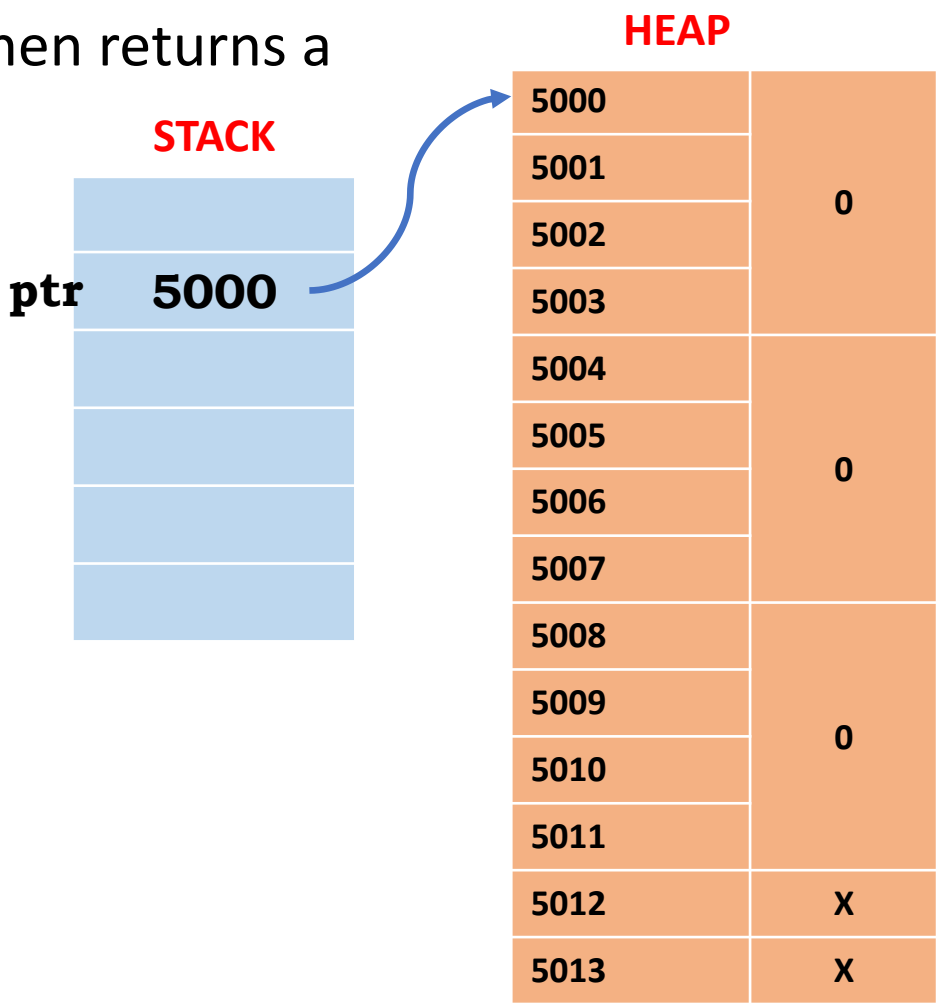
//allocates memory for an array of nmemb elements of size bytes each

- **Example**:

```
int* ptr = (int*) calloc (3,sizeof (int));
```

//Allocating memory for an array of 3 elements of integer type

- Coding example



PES UNIVERSITY

realloc() - reallocation of memory

- Modifies the size of previously allocated memory using malloc or calloc functions
- Returns a pointer to the newly allocated memory which has the new specified size. Returns NULL for an unsuccessful operation
- If realloc() fails, the original block is left untouched
- **Syntax:** void *realloc(void *ptr, size_t size);
- If ptr is NULL, then the call is equivalent to malloc(size), for all values of size
- If size is equal to zero, and ptr is not NULL, then the call is equivalent to free(ptr)
- This function can be used only for dynamically allocated memory, else behavior is undefined

PES UNIVERSITY

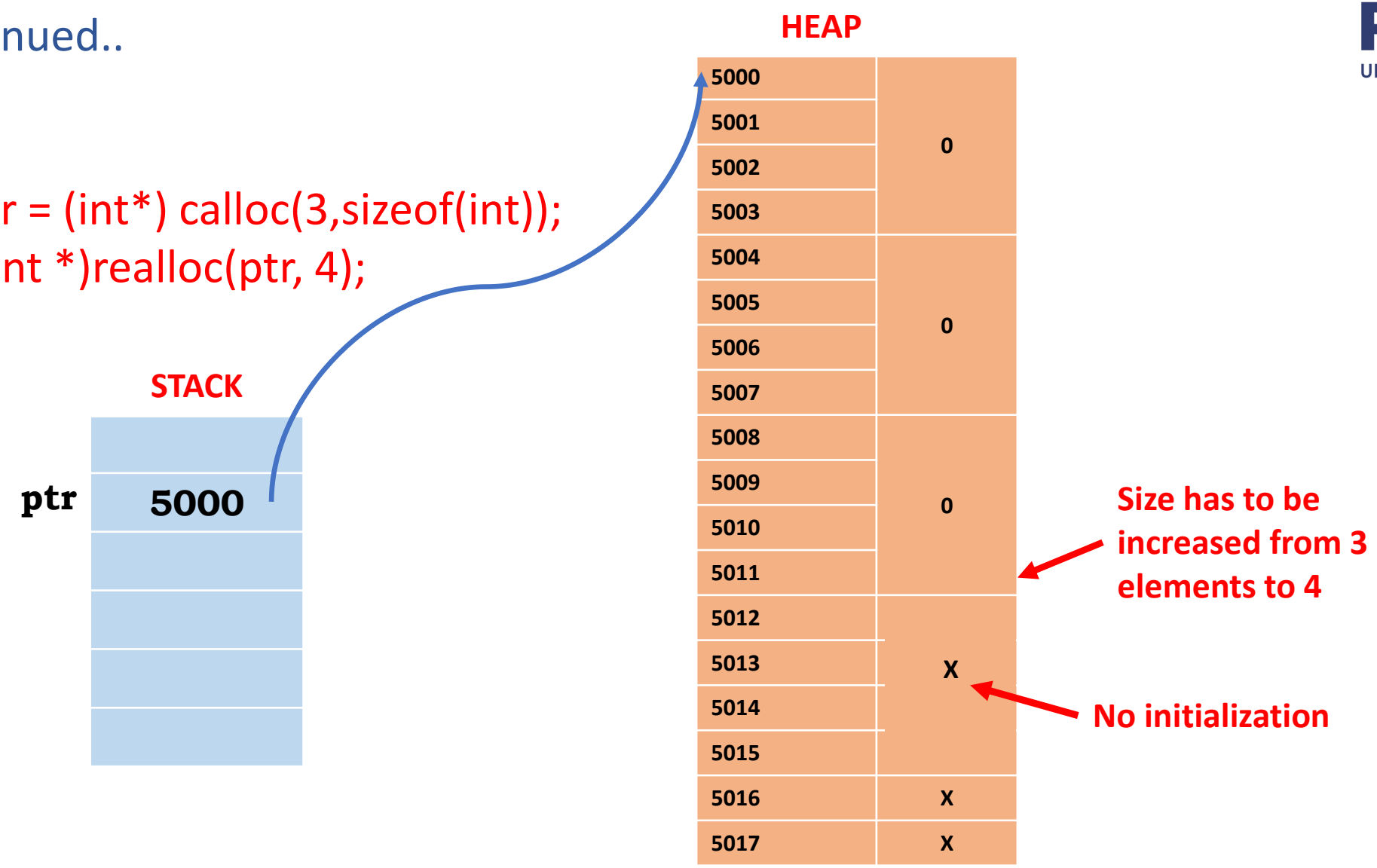
realloc() continued..

- The content of the memory block is preserved up to the lesser of the new and old sizes, even if the block is moved to a new location
- If the new size is larger than the old size, then it checks if there is an option to expand or not.
  - If the existing allocation of memory can be extended, it extends it but the added memory will not be initialized.
  - If memory cannot be extended, a new sized memory is allocated, initialized with the same older elements and pointer to this new address is returned. Here also added memory is uninitialized.
- If the new size is lesser than the old size, content of the memory block is preserved.
- Coding examples

realloc() continued..

Example:

```
int* ptr = (int*) calloc(3,sizeof(int));
ptr = (int *)realloc(ptr, 4);
```



free()

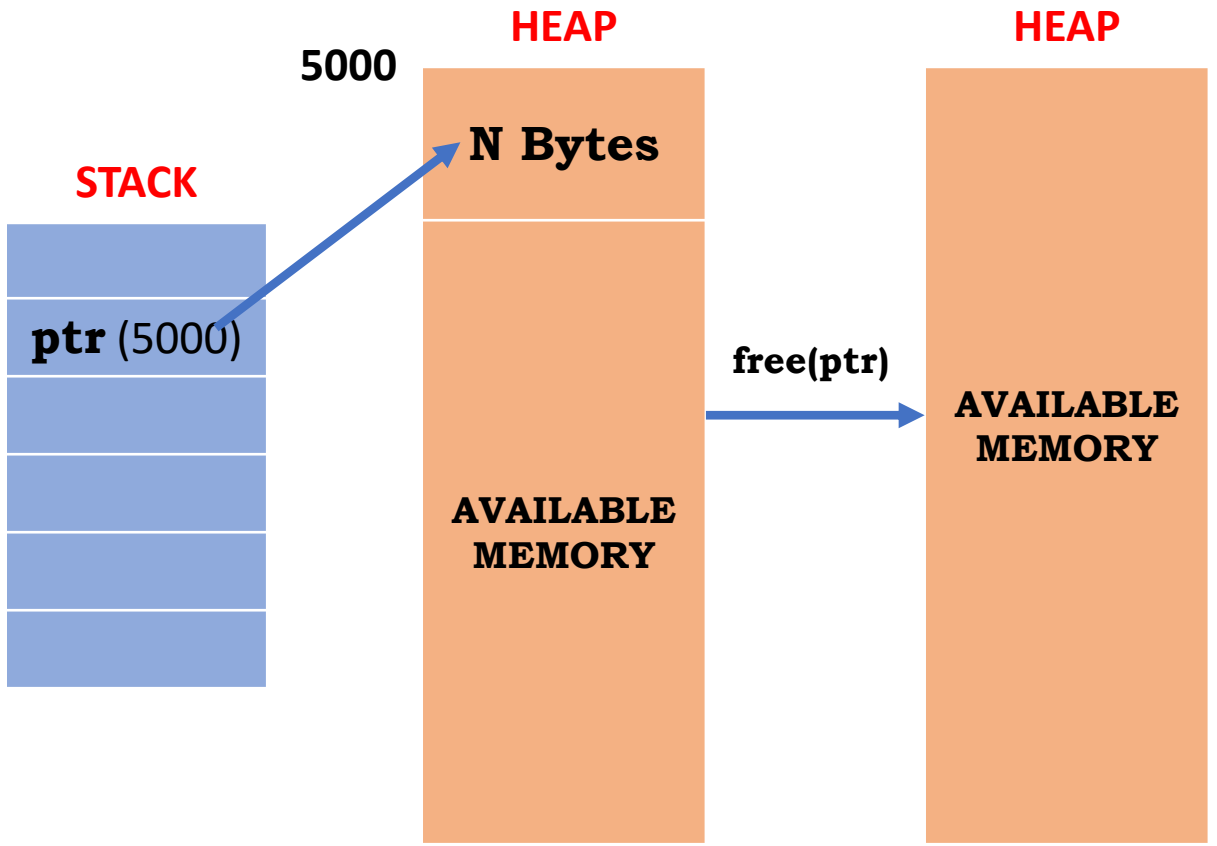
- Releases the allocated memory and returns it back to heap

- **Syntax**:

  free (ptr); //ptr is a pointer to a memory block which has been previously created using malloc/calloc

- No size needs to be mentioned in the free().

- On allocation of memory, the number of bytes allocated is stored somewhere in the memory. This is known as **book keeping information**



THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CCBD & CDSAML, PESU
Prof. Sindhu R Pai - sindhurpai@pes.edu
Dr. Shruti Jadon, CSE, PESU

Ack: Teaching Assistant - U Shivakumar