Problem Solving With C - UE24CS151B

Strings in C

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

PROBLEM SOLVING WITH C

Strings in C

1. Introduction
2. Declaration
3. Initialization
4. Demo of C Code
5. String v/s Pointer

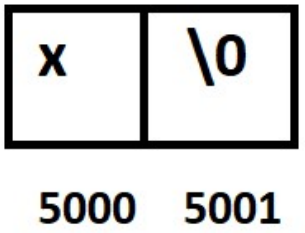
PES UNIVERSITY

Introduction

- An array of characters and terminated with a special character '\0' or NULL. ASCII value of NULL character is 0.

- String constants are always enclosed in double quotes. It occupies one byte more to store the null character.

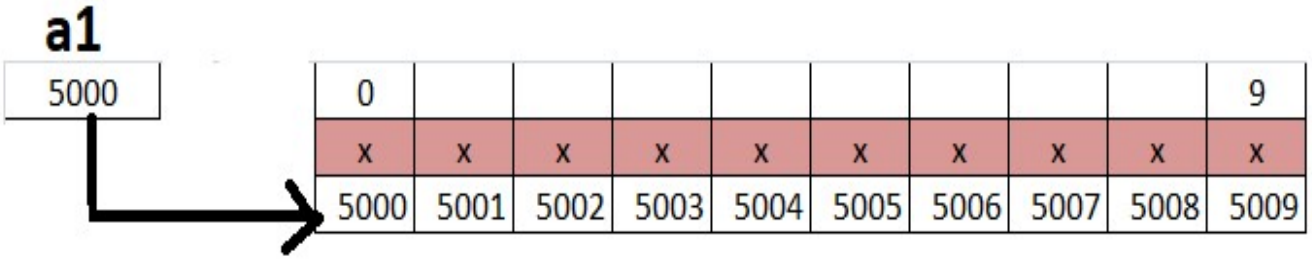
- Example: **"X" is a String constant**



Declaration

Syntax: **char variable_name[size];** //Size is compulsory

```
char a1[10];        // valid declaration
char a1[];          // invalid declaration
```

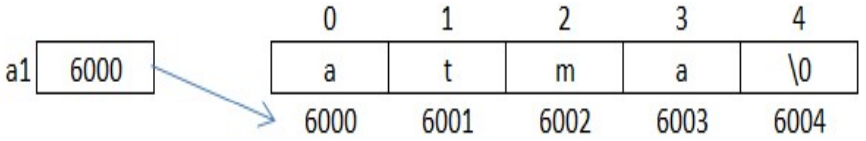


Initialization

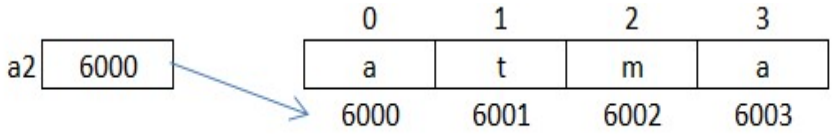
Syntax: **char variable_name[size] = {Elements separated by comma};**

Version 1:

- char a1[] = {'a', 't', 'm', 'a', '\0' };
- Shorthand notation: char a1[] = ”atma”;

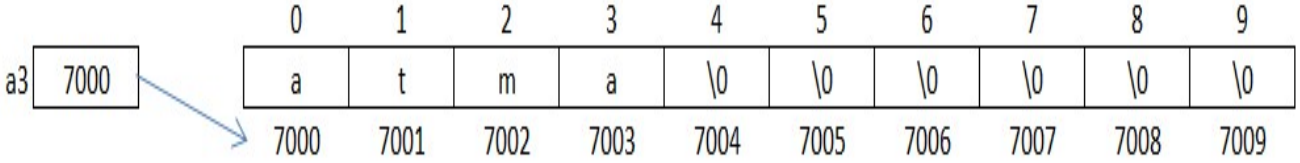


Version 2: char a2[] = {'a', 't', 'm', 'a' };



Version 3: Partial initialization

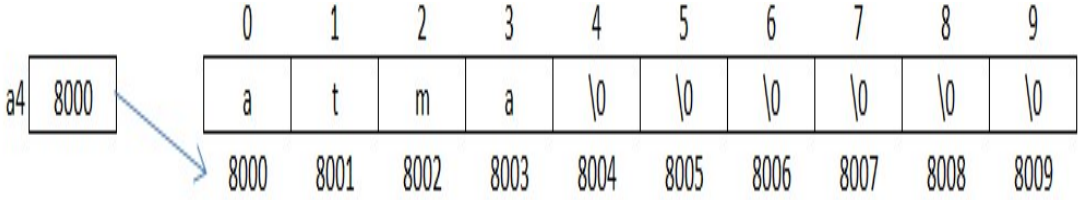
- char a3[10] = {'a','t','m','a'};



Initialization continued..

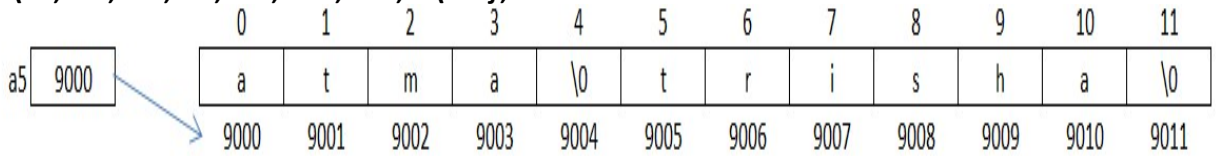
Version 4: Partial initialization

- char a4[10] = {'a','t','m','a', '\0'};
- char a4[10] = ”atma”;

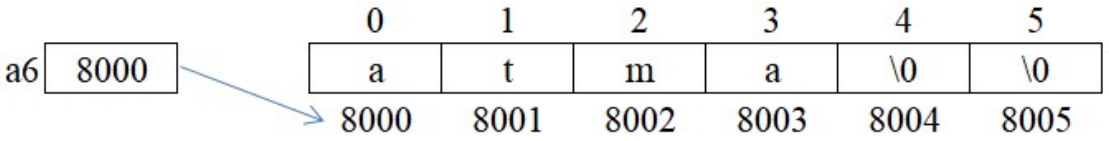


Version 5: char a5[ ] = {'a', 't', 'm', 'a', '\0', 't', 'r', 'i', 's', 'h', 'a', '\0' };

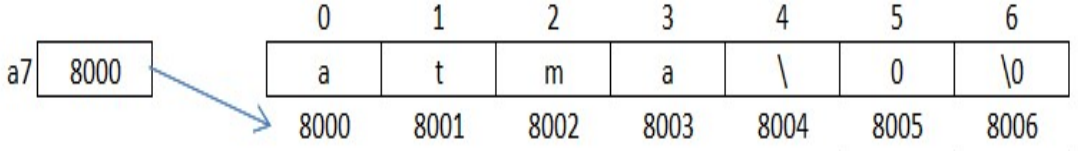
Version 6: char a6[ ] = "atma\0" ;



Version 7: char a7[ ] = "atma\\0" ;



Version 8: char a8[ ] = "at\0ma" ;



Demo of C Code

- To read and display a string in C
- Points to note:
  - If the string is hard coded, it is programmer's responsibility to end the string with '\0' character.
  - scanf terminates when white space is given in the user input.
  - scanf with %s will introduce '\0' character at the end of the string. printf with %s requires the address and will display the characters until '\0' character is encountered
  - If you want to store the entire input from the user until user presses new line in one character array variable, use [^\n] with %s in scanf

PES UNIVERSITY

String v/s Pointer

- char x[] = "pes"; // x is an array of 4 characters with ‘p’, ‘e’, ‘s’, ‘\0
  Stored in the Stack segment of memory
- Can change the elements of x. x[0] = ‘P’;
- Can not increment x as it is an array name. x is a constant pointer.

- char *y = “pes”;
  y is stored at stack. “pes” is stored at code segment of memory. It is read only.
- y[0] = ‘P’ ; // undefined behaviour
- Can increment y . y is a pointer variable.

THANK YOU

Department of Computer Science and Engineering

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

Problem Solving With C - UE24CS151B

String manipulation Functions & Errors

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

PROBLEM SOLVING WITH C

String manipulation Functions & Errors

1. Builtin String manipulation functions
2. User defined string functions
3. Error demonstration

Built-in String manipulation Functions

- Expect '\0' and available in string.h
- In character array, if '\0' is not available at the end, result is undefined when passed to built-in string functions

- **strlen(a)** – Expects string as an argument and returns the length of the string, excluding the NULL character
- **strcpy(a,b)** – Expects two strings and copies the value of b to a.
- **strcat(a,b)** – Expects two strings and concatenated result is copied back to a.
- **strchr(a,ch)** – Expects string and a character to be found in string. Returns the address of the matched character in a given string if found. Else, NULL is returned.
- **strcmp(a,b)** – Compares whether content of array a is same as array b. If a==b, returns 0. Returns 1, if array a has lexicographically higher value than b. Else, -1.

PES UNIVERSITY

User defined String functions

- Start with iterative solution
- Usage of pointer arithmetic operations
- Usage of recursion to get the solution for few of the below functions

1. my_strlen()
2. my_strcpy()
3. my_strcmp()
4. my_strchr()
5. my_strcat()

PES UNIVERSITY

Error Demonstration

1. char * name[10] ; // declaring an array of 10 pointer pointing each one to a char

   char name[10];

2. name = “choco”; // Cannot assign a string by using the operator =

   char name[10] = “choco“ OR use strcpy()

3. printf("%s\n", *name);

   Provide only the address of the first element of the string

4. Coding examples to demo the error caused while applying built-in string functions on strings without specifying ‘\0’ at the end of the string

THANK YOU

Department of Computer Science and Engineering

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

Ack: Teaching Assistant - U Shivakumar

Problem Solving With C - UE24CS151B

Command Line Arguments

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

PROBLEM SOLVING WITH C

Command Line Arguments

PES UNIVERSITY

- Introduction
- Usage of argc and argv
- Demo of C Code

Command Line Arguments

Introduction

- Providing data to the program whenever the command to execute is used

- Provided after the name of the executable in command-line shell of Operating Systems

  Example: **a.exe 18 27 //** a.exe is the executable for the code to add two numbers

  // 8 27 are numbers to be added

- All the arguments which are passed in the command line are accessed as **strings** inside the program .

- Use **atoi** function to convert to integer if integers are passed in Command line

PES UNIVERSITY

Usage of Argument count(argc) and Argument vector(argv)

- To pass command line arguments, define main() with two arguments

  int main (int argc , char *argv []) { … }

- **argc:** An integer which specifies the number of arguments passed in the command line including the executable. The value of argc should be Non- Negative

  Example: a.exe 18 27 (argc=3)

- **char * argv[]:** An array of character pointers which contains all the arguments passed in the command line

  Example: argv[0] is a.exe

  argv[1] is 18

PES UNIVERSITY

Demo of C Code

- Find the sum of all numbers provided in the command line

THANK YOU

Department of Computer Science and Engineering

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

Ack: Teaching Assistant - U Shivakumar

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];)

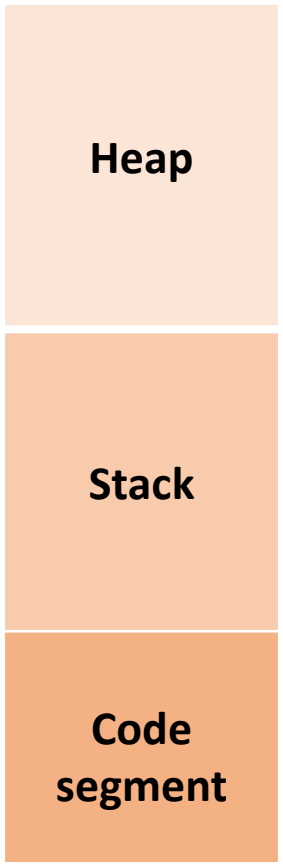
2. **Automatic allocation**
   - decided by the compiler
   - allocation at run time
   - allocation on entry to the block and deallocation on exit
   - example: function call (stack space is used and released as soon as callee function returns back to the calling function)

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

PES UNIVERSITY

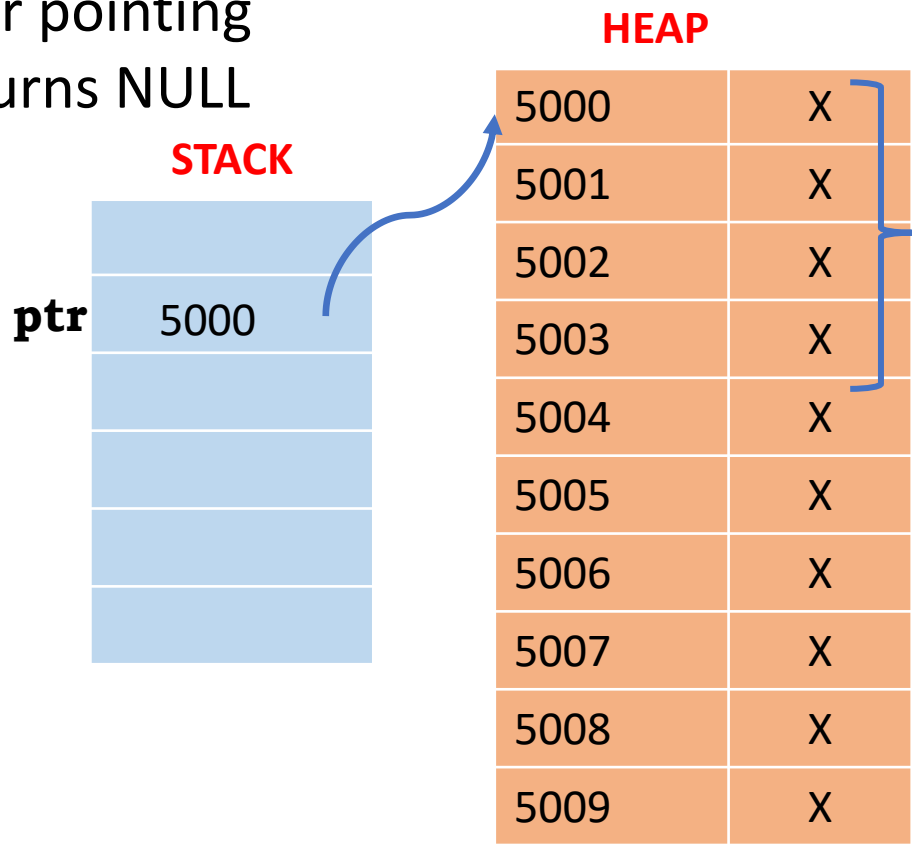
Dynamic Allocation

- 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);
```

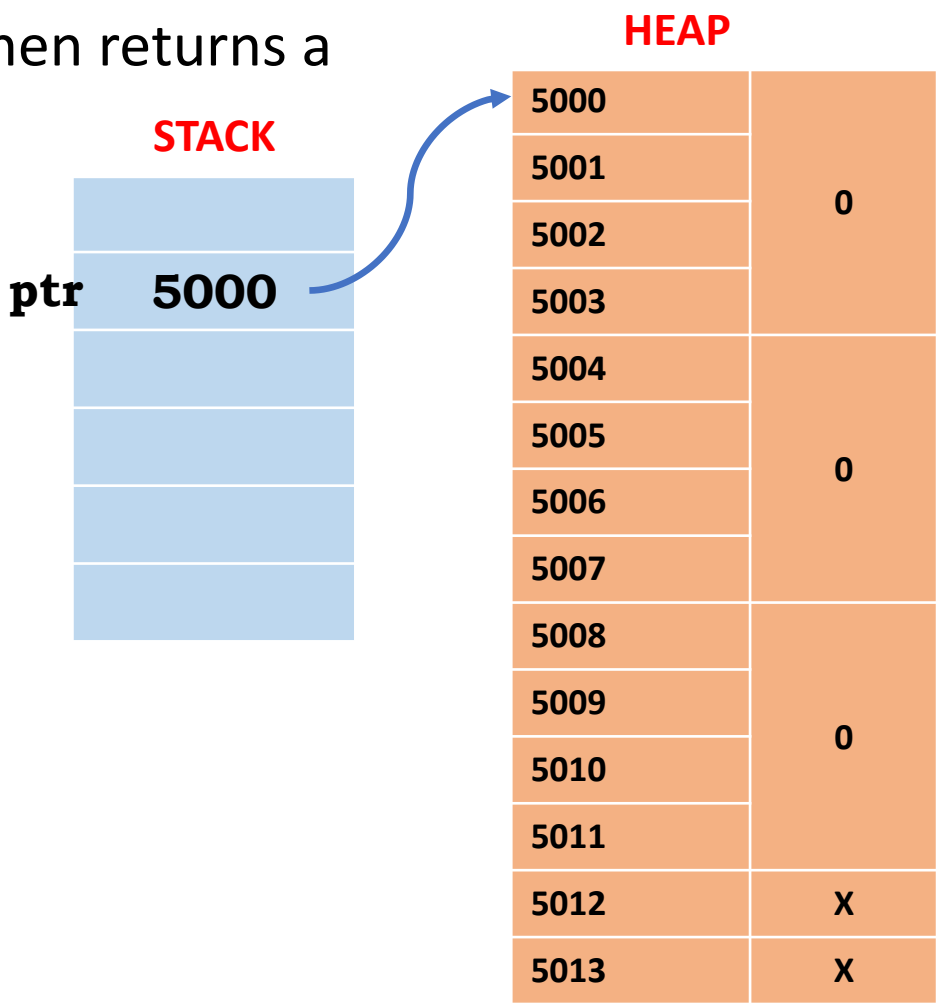
//allocates memory for an array of nmemb elements of size bytes each

- **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

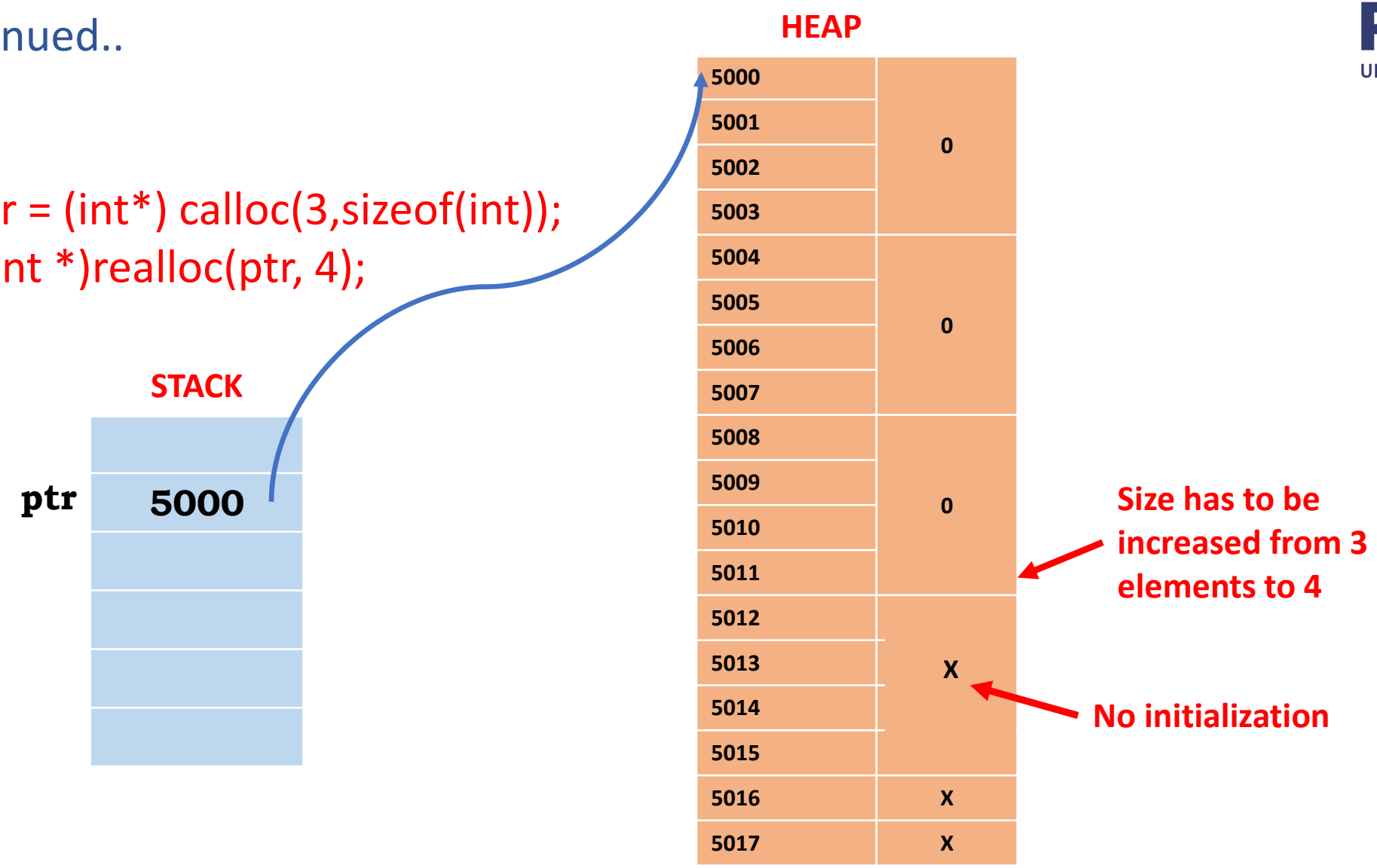
realloc() continued..

- 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()

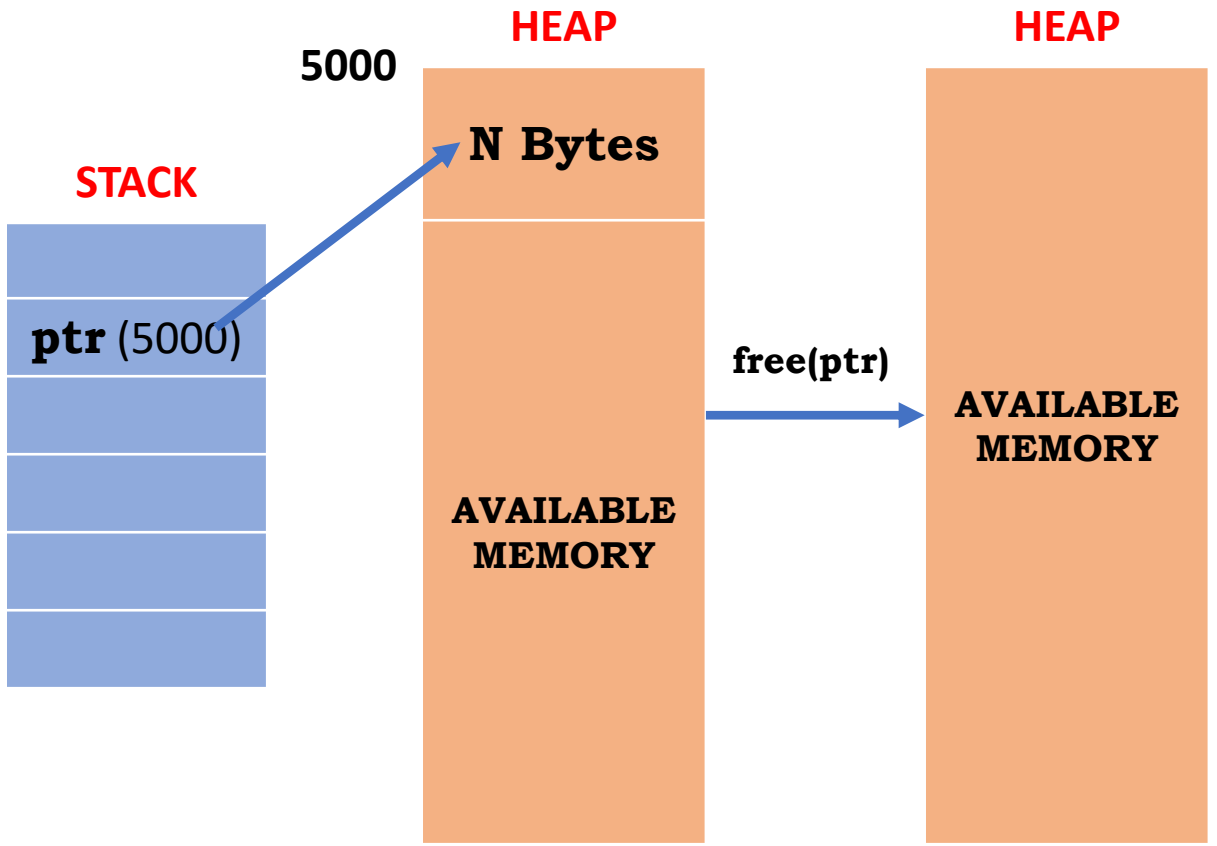
- Releases the allocated memory and returns it back to heap

- **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

Problem Solving With C - UE24CS151B

Structures in C

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

Structures in C

PES UNIVERSITY

- Introduction
- Characteristics
- Declaration
- Accessing members
- Initialization
- Memory allocation
- Comparison

PES UNIVERSITY

Introduction

- A user-defined data type that allows us to combine data of different types together.
- Helps to construct a complex data type which is more meaningful.
- Provides a single name to refer to a collection of related items of different types.
- Provides a way of storing many different values in variables of potentially different types under the same name.
- Generally useful whenever a lot of data needs to be grouped together.
- Creating a new type decides the binary layout of the type

PES UNIVERSITY

Characteristics/Properties

- Contains one or more components(homogeneous or heterogeneous) – Generally known as data members. These are named ones.
- **Order of fields and the total size of a variable** of that type is decided when the new type is created
- Size of a structure depends on implementation. Memory allocation would be **at least equal to the sum of the sizes of all the data members** in a structure. Offset is decided at compile time.
- Compatible structures may be assigned to each other.

Introduction

Syntax :

- Keyword **struct** is used for creating a structure.

- The format for declaring a structure is as below:

```
struct <structure_name>
{
        data_type member1;
        data_type member2;
        .....
        data_type memebern;
};          // semicolon compulsory
```

Example: User defined type Student entity is created.

```
struct Student
{
        int roll_no;
        char name[20];
        int marks;
};
```

Note: No memory allocation for declaration/description of the structure.

PES UNIVERSITY

Declaration

- Members of a structure can be accessed only when instance variables are created

- If struct Student is the type, the instance variable can be created as:

```
struct student s1;        // s1 is the instance variable of type struct Student

struct student* s2;       // s2 is the instance variable of type struct student*.
                          // s2 is pointer to structure
```

- Declaration (global) can also be done just after structure body but before semicolon.

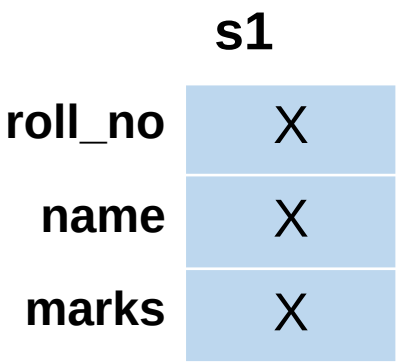


Fig. 1. After declaration, only undefined entries (X)

PES UNIVERSITY

Initialization

- Structure members can be initialized using curly braces '{}' and separated by comma.
- Data provided during initialization is mapped to its corresponding members by the compiler automatically.
- Further extension of initializations can be:

1. **Partial initialization**: Few values are provided.
   Remaining are mapped to zero. For strings, '\0'.
2. **Designated initialization**:
   - Allows structure members to be initialized in any order.
   - This feature has been added in C99 standard.
   - Specify the name of a field to initialize with **'.member_name =' OR**
     'member_name:' before the element value. Others are initialized to default value.

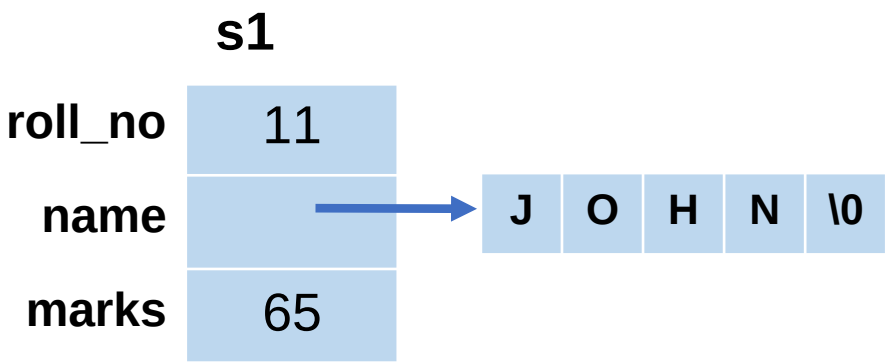


Fig. 2. After initialization, entries are mapped

PES UNIVERSITY

Accessing data members

- Operators used for accessing the members of a structure.
  1. **Dot operator (.)**
  2. **Arrow operator (->)**
- Any member of a structure can be accessed using the structure variable as:

structure_variable_name.member_name

Example: s1.roll_no

//where s1 is the structure variable name and roll_no member is data member of s1.

- Any member of a structure can be accessed using the pointer to a structure as:

pointer_variable->member_name

Example: s2->roll_no

// where s2 is the pointer to structure variable and we want to access roll_no member of s2.

PES UNIVERSITY

Memory allocation

- **At least equal to the sum of the sizes of all the data members.**
- Size of data members is implementation specific.
- Coding Examples

PES UNIVERSITY

Comparison of structures

- Comparing structures in C is not permitted to check or compare directly with logical and relational operators.
- Only structure members can be compared with relational operator.
- Coding examples

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

Problem Solving With C - UE24CS151B

Structures in C

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

Structures in C

- Member-wise copy
- typedef
- Nested structures
- Passing structure to functions

PES UNIVERSITY

Member-wise copy

- Structures of same type are assignment compatible.
- When you assign one structure variable to another structure variable of same type, member- wise copy happens.
- All structure members with values (if initialized) are copied
- Both copies does not point to the same memory location.
- Any change in one copy will not be reflected in the other.
- Coding examples

typedef

- Allows users to provide alternative names for the primitive (e.g., int, float etc) and user-defined (e.g struct) data types.
- Only adds a new name for some existing data type but does not create a new type.
- Syntax typedef <existing_datatype> <new_name>;

Example:

```
//without typedef
int a, b ;

//with typedef
typedef int integer;
integer a, b ;
```

- Coding examples wrt structures

PES UNIVERSITY

Nested Structures

- Structure written inside another structure is called as nesting of two structures.
- Two ways

WAY 1 : Declare two separate structures and using dependent structure inside the main structure as a member

WAY 2 : Declare embedded structures

- Coding examples

Passing structure to a function

- Parameter passing is always by value.
- Argument is copied to parameter and modifications inside the function body applies to parameter only.
- If you want to change the argument, pass a pointer to a structure(l-value) as an argument.
- C code demonstration : To read and display a structure

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

Problem Solving With C - UE24CS151B

Array of Structures

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

PROBLEM SOLVING WITH C

Array of Structures

1. Introduction
2. Array of Structure Variable Definition
3. Array of Structure Variable Initialization
4. Pointer to an Array of Structures

Array of Structures

Introduction

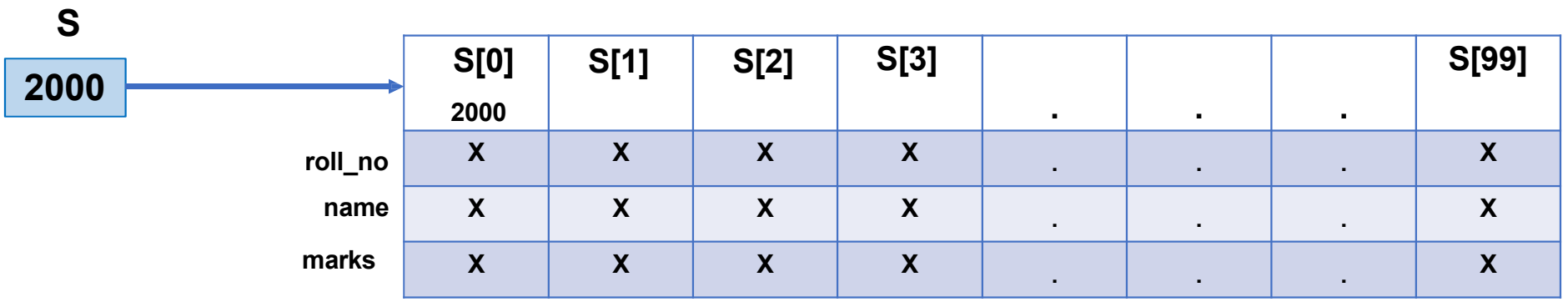
- Think about storing the roll number, name and marks of one student

  We need structures to store different types of related data.

  struct student s;
- Think about storing the roll number, name and marks of 100 students

  We need an Array of structures

  struct student S[100];
- S[0] stores the information of first student, S[1] stores the information of second student and so on.



Declaration of Structure variable

- Can be done in two ways
  - Along with structure declaration after closing } before semicolon (;) of structure body.

```
struct student {
        int roll_num; char name[100]; int marks;
}s[100];
```

  - After structure declaration (either inside main or globally using struct keyword)

```
struct student s[100];
```

- Coding examples

PES UNIVERSITY

Initialization of structure variable

- **Compile time initialization:** Using a brace-enclosed initializer list

```
struct student S[] = { {1, “John”, 60}, {2,”Jack”, 40}, {3, “Jill”, 77} };
struct student S[3] = { {11, “Joseph”, 60}}; //partial initialization
struct student S[2] = {1, “John”, 60, 2,”Jack”, 40};
```

- **Run time initialization (**using loops preferably**)**

- Coding examples

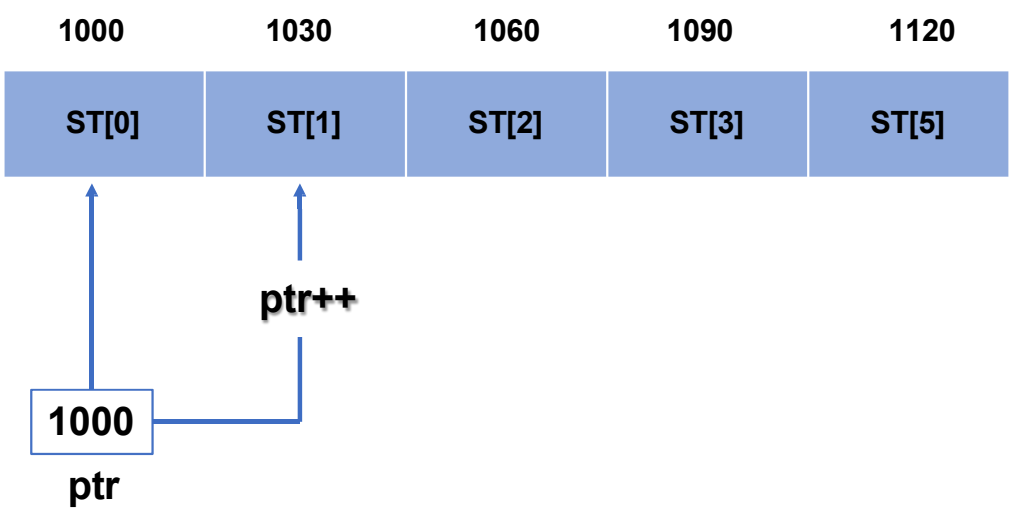
Array of Structures

Pointer to an Array of Structures

- Used to access the array of structure variables efficiently

```
struct student
{       int roll_no;
        char name[22];
        int marks;
}ST[5];

struct student *ptr = ST;
```



- Coding examples to print the array of structures using pointer

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

Problem Solving With C - UE24CS151B

Bit fields in C

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

PROBLEM SOLVING WITH C

Bit fields in C

- What is a Bit field?
- Bit field creation
- Few points wrt Bit fields

Bit fields in C

What is a Bit field ?

- The variables defined with a predefined width and can hold more than a single bit
- Consists of a number of adjacent computer memory locations which have been allocated to hold a sequence of bits.
- The meaning of the individual bits within the field is determined by the programmer

- **Great significance in C programming** because of the following reasons
  - Used to reduce memory consumption
  - Easy to implement
  - Provides flexibility to the code

Bit fields in C

Bit field Creation

- **Syntax:** struct [tag] { type [member_name] : width ; };

  type - Determines how a bit-field's value is interpreted. May be int, signed int, or unsigned int

  member_name - The name of the bit-field

  width - Number of bits in the bit-field. The width must be less than or equal to the bit width of the specified type. The largest value that can be stored is 2^n -1, where n is bit-length

- **Example:** struct Status {

  unsigned int bin1:1; // 1 bit is allocated for bin1. only two digits can be stored 0 and 1

  unsigned int bin2:1; // if it is signed int bin1:1 or int bin1:1, one bit is used to represent the sign

  };

- Coding examples

Few points wrt Bit fields

- The first field always starts with the first bit of the word. Cannot overlap integer boundaries
- Cannot extract the address of bit field
- Should be assigned values that are within the range of their size. It is implementation defined to assign an out-of-range value to a bit field member
- Cannot have pointers to bit field members as they may not start at a byte boundary
- Array of bit fields not allowed
- Storage class cannot be used on bit fields
- Can use bit fields in union
- Unnamed bit fields results in forceful alignment of boundary

THANK YOU

Department of Computer Science and Engineering

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

Problem Solving With C - UE24CS151B

Unions in C

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

PROBLEM SOLVING WITH C

Unions in C

- What is Union?
- Accessing Union Members
- Union vs Structure

Unions in C

What is Union?

- A user defined data type which may hold members of different sizes and types
- Allow data members which are **mutually exclusive to share the same memory**
- Unions provide an efficient way of using the same memory location for multiple-purpose
  - At a given point in time, only one can exist
- The memory occupied will be large enough to hold the largest member of the union
  - **The size of a union is the size of the biggest component**
- All the **fields overlap** and they have the **same offset : 0**.
- Used while coding embedded devices

Accessing the Union members

- **Syntax:**

```
union Tag          // union keyword is used
{
data_type member1;     data_type member2;        … data_type member n;
};          // ; is compulsory
```

- To access any member using the variable of union type,
  - use the **Member Access operator (.)**
- To access any member using the variable of pointer to union type,
  - use the **Arrow operator (->)**
- Coding Examples

Unions in C

PES UNIVERSITY

Union Vs Structure

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

THANK YOU

Department of Computer Science and Engineering

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

Problem Solving With C - UE24CS151B

Enumerations

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

PROBLEM SOLVING WITH C

Enumerations

- Introduction
- Enum creation
- Points wrt Enums
- Demo of Enums in C

Enumerations

Introduction

- A way of creating user defined data type to assign names to integral constants. Easy to remember names rather than numbers

- Provides a symbolic name to represent one state out of a list of states

- The names are symbols for integer constants, which won't be stored anywhere in program's memory

- Used to **replace #define chains**

PES UNIVERSITY

Enum creation

- **Syntax:**

```
enum identifier { enumerator-list };        // semicolon compulsory
                                            // identifier optional
```

- Example:

```
enum Error_list { SUCCESS, ERROR, RUN_TIME_ERROR, BIG_ERROR };
```

- Coding Examples

Points wrt Enums

- Enum names are automatically assigned values if no value specified
- We can assign values to some of the symbol names in any order. All unassigned names get value as value of previous name plus one.
- Only integer constants are allowed. Arithmetic operations allowed-> + , -, *, / and %
- Enumerated Types are Not Strings. Two enum symbols/names can have same value
- All enum constants must be unique in their scope. It is not possible to change the constants
- Storing the symbol of one enum type in another enum variable is allowed
- One of the short comings of Enumerated Types is that they don't print nicely

Domo of Enum in C

- Demo of Enum points discussed in the previous slide

THANK YOU

Department of Computer Science and Engineering

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

Ack: Teaching Assistant - U Shivakumar

Problem Solving With C - UE24CS151B

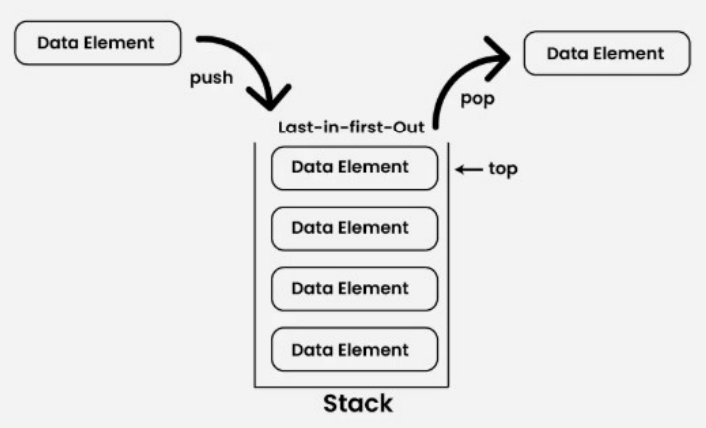
Stack

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

Stack

What is a stack?

- Data structure that stores data using the **Last In First Out (LIFO)** principle
- Imagine a stack of books. The last book you put on top of the stack (LI) is the first one you take off the stack (FO)
- Linear data structure: Stores data linearly or sequentially

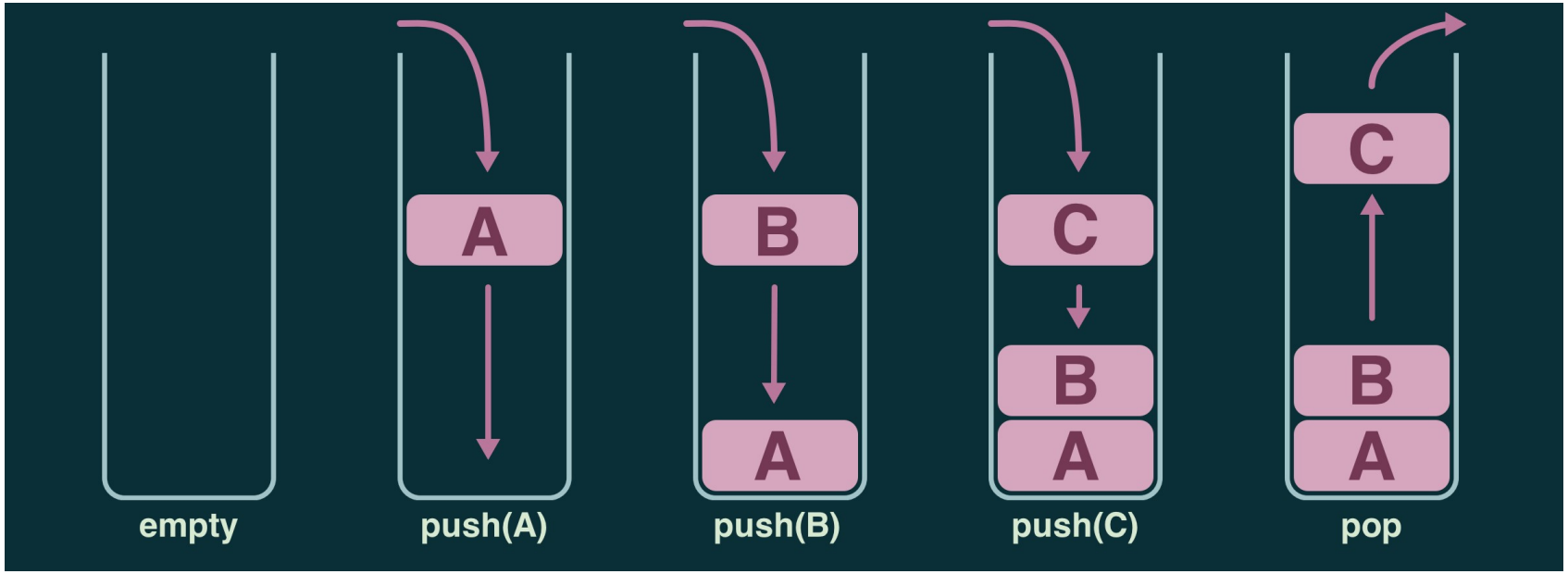


PES UNIVERSITY

Key Terms and Operations

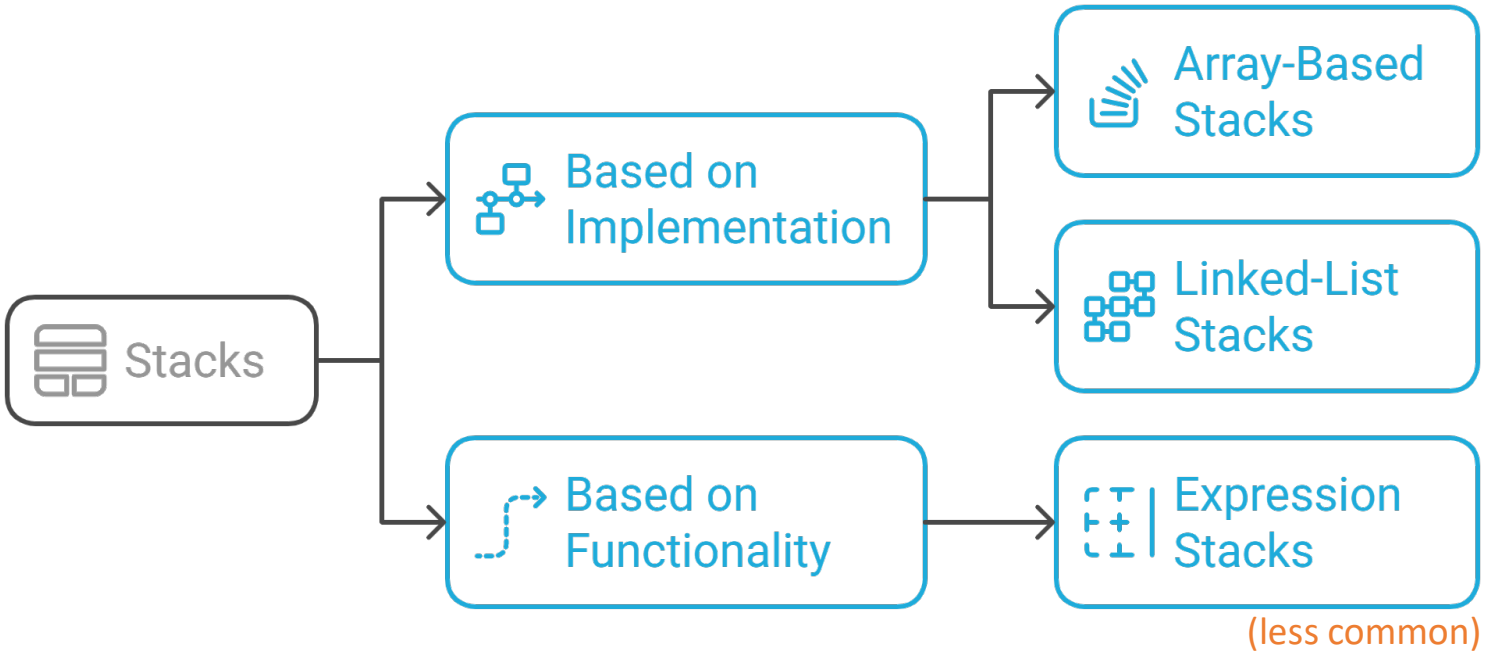
- **Top:** The end of the stack where modifications are made
- **Push:** Adds an element to the top of the stack.
- **Pop:** Removes the element from the top of the stack.
- **Peek:** Returns the top element without removing it.
- **IsEmpty:** Checks if the stack is empty.
- **IsFull:** Checks if the stack is full (for fixed-size stacks).

Stack



PES UNIVERSITY

Types of Stacks



PES UNIVERSITY

Array Based Stack

- **Data Structure:** Array of fixed size

- **Variables:** Top (index of the top element, usually initialized to -1)

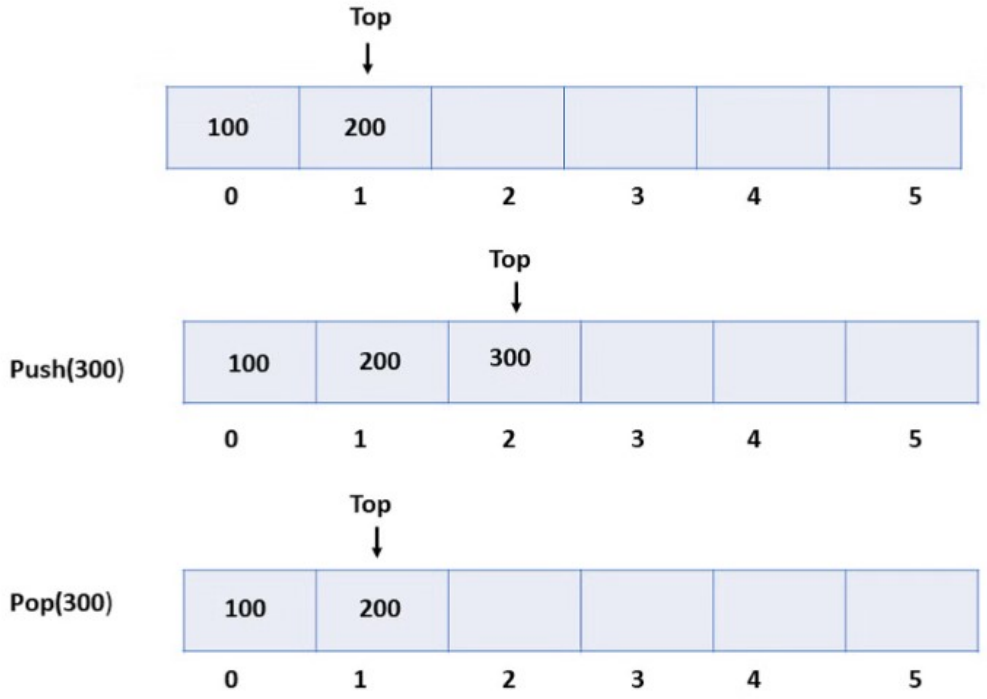
- **Operations:**
  - Push: Increment top, add element at array[top]
    - Check for overflow (IsFull condition)

  - Pop: Return element at array[top], decrement top
    - Check for underflow (IsEmpty condition)

  - Peek: Return element at array[top] without modifying top

PES UNIVERSITY

Array Based Stack



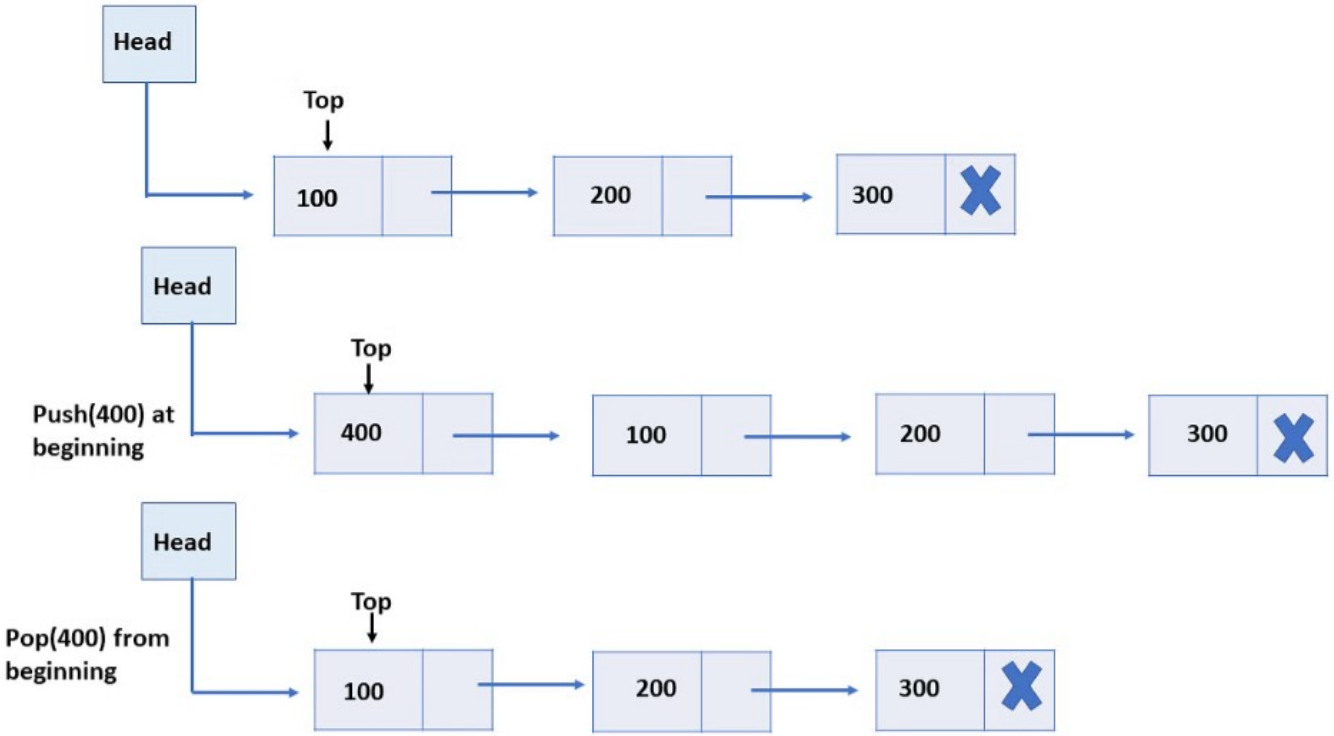
PES UNIVERSITY

Linked List Stack

- **Data Structure:** Singly linked list
- **Variables:** Top (pointer to the top node)
- **Operations:**
  - Push: Create a new node, set its next to top, update top to the new node.
  - Pop: Store the top node's data, update top to top.next, free the old top node.
  - Peek: Return the data of the top node.
  - IsEmpty: Check if top is NULL/None.

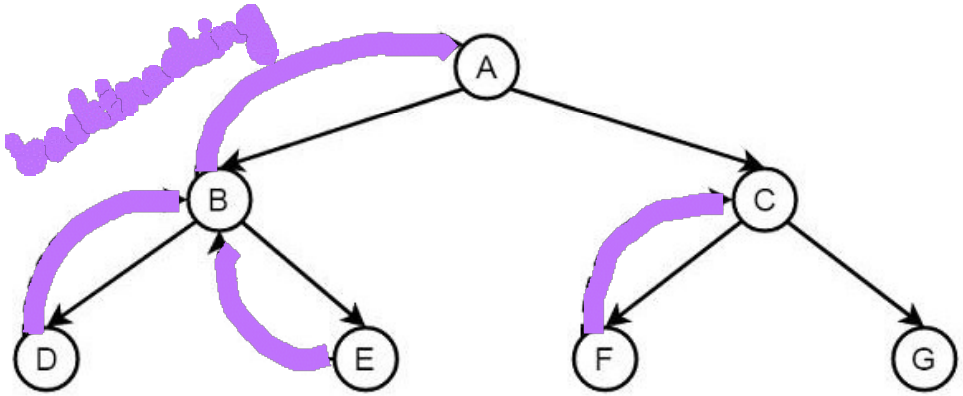
PES UNIVERSITY

Linked List Stack



Applications

• **Backtracking** is implemented using stacks. This is a recursive algorithm that is used to solve optimization problems

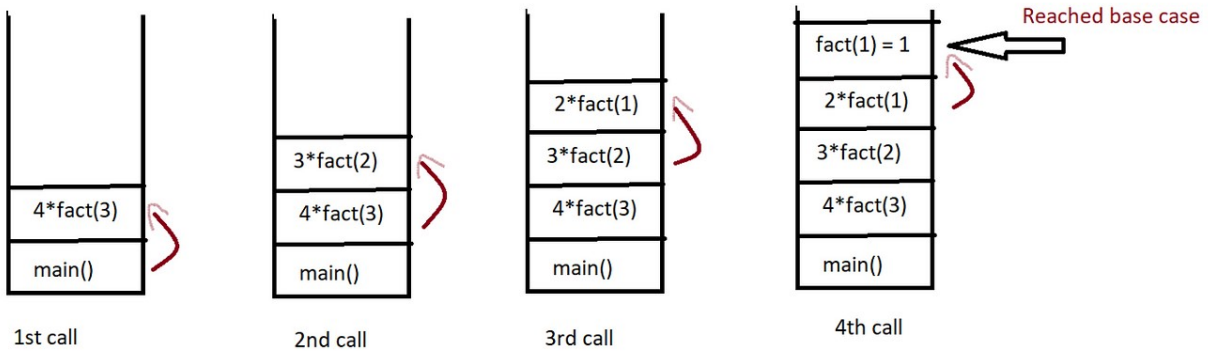


• Stacks are also used to **evaluate expressions written in postfix/prefix notations**, and to convert infix to postfix/prefix (this will be covered in a later course).

Applications

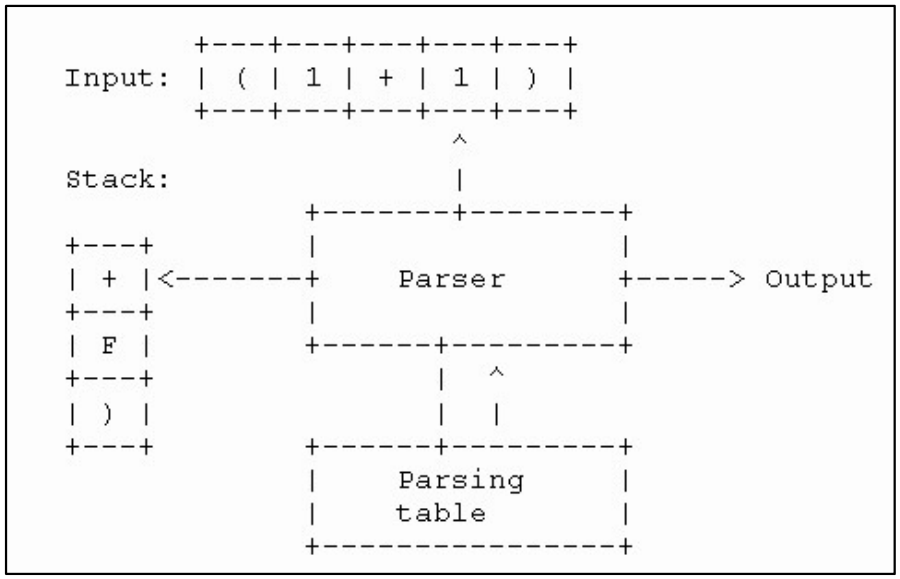
• **Function calls are kept track of** using stacks in the computer system. Every call results in a record being added to the stack, which is then popped once the function returns.

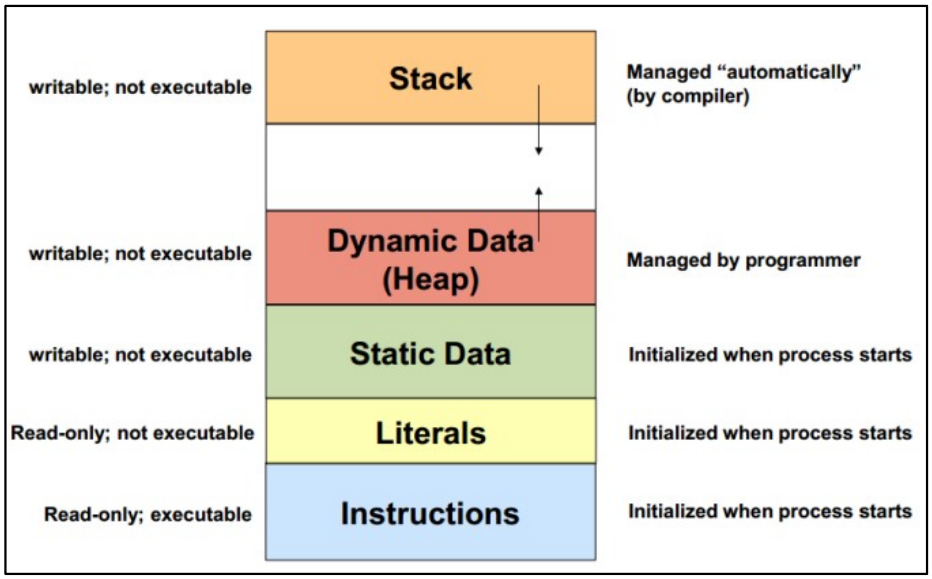
• The above naturally extends to the concept of **recursion** as well; this is also kept track of using stacks.



Applications

Stacks are also used for **syntax parsing** in compilers and **memory management** in operating systems(For further exploration)





THANK YOU

Department of Computer Science and Engineering

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

Ack: Teaching Assistant – Advaith Sanil Kumar

Problem Solving With C - UE24CS151B

List

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

PROBLEM SOLVING WITH C

List

1. Introduction
2. Self Referential Structures
3. Characteristics
4. Pictorial Representation
5. Operations
6. Different Types
7. Applications

List

Introduction

- Collection of nodes connected via links.
- The node and link has a special meaning in C.
- Few points to think!!

  - Can we have pointer data member inside a structure? - Yes

  - Can we have structure variable inside another structure? - Yes

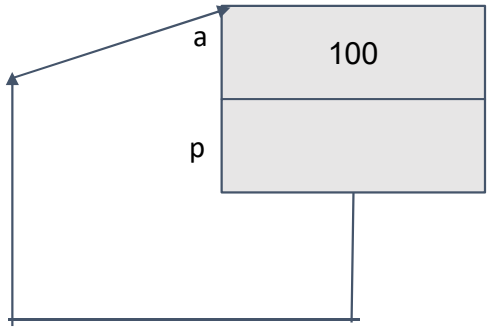
  - Can we have structure variable inside the same structure ? – No

    Solution is: Have a pointer of same type inside the structure

Self Referential Structures

- A structure which has a pointer to itself as a data member

```
struct node
{
        int a;
        struct node *p;
} ; // defines a type struct node
// memory not allocated for type declaration
```



- Variable declaration to allocate memory and assigning values to data members

```
struct node s;    s.a = 100;          s.p = &s;
```

List

Characteristics

- A data structure which consists of zero or more nodes.
- Every node is composed of two fields: data/component field and a pointer field
- The pointer field of every node points to the next node in the sequence
- Accessing the the nodes is always one after the other. There is no way to access the node directly as random access is not possible in linked list. Lists have sequential access
- Insertion and deletion in a list at a given position requires no shifting of element

Operations on List

- Insertion
- Deletion
- Search
- Display
- Merge
- Concatenate
- ..

Pictorial Representation

- Structure definition of a node

```
struct node {
        int info; // component field
        struct node *link; // pointer field
};
typedef struct node NODE_T;
```

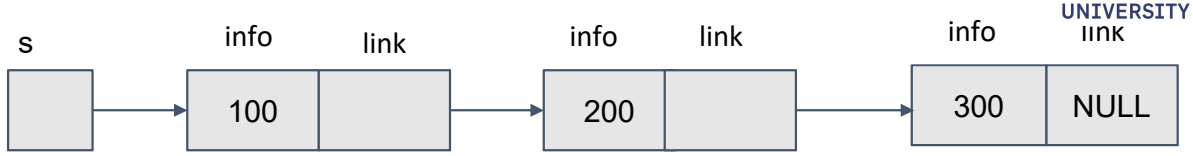


Fig1: Linked list Representation

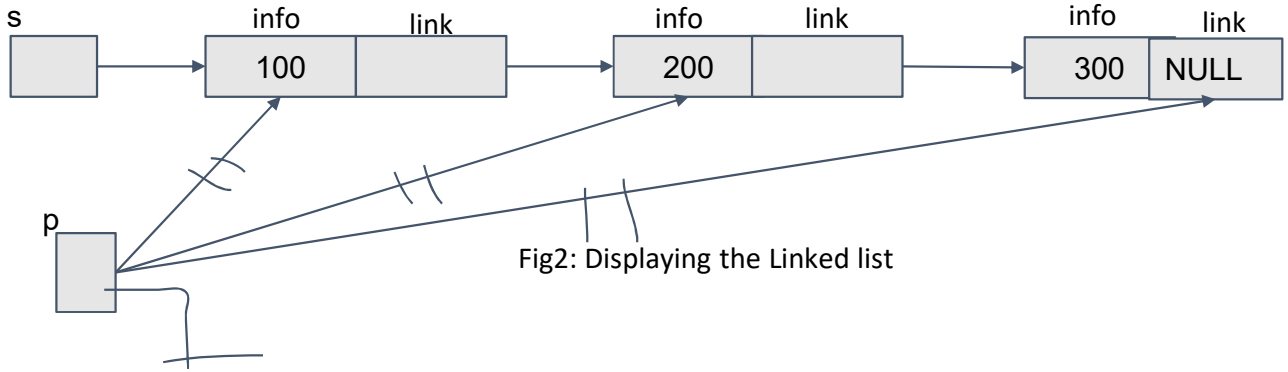


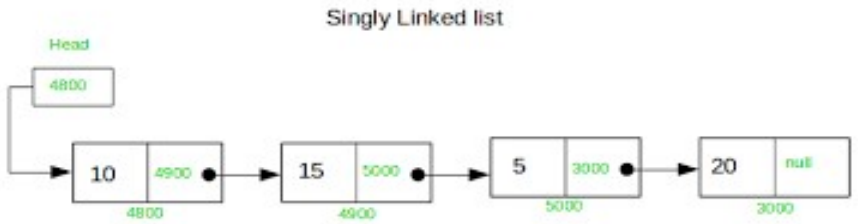
Fig2: Displaying the Linked list

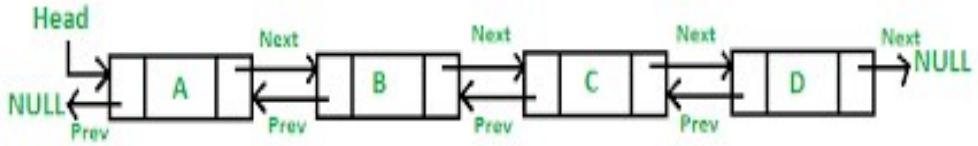
- Demo of C code to create, display and free the list shown in pictorial representation

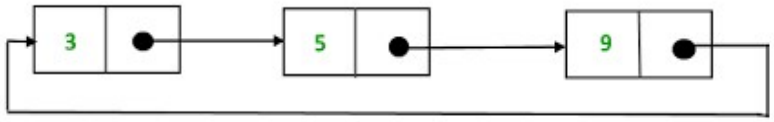
PES UNIVERSITY

Different Types

- Singly Linked List
- Doubly Linked List
- Circular Linked List







Applications

- Implementation of Stacks & Queues
- Implementation of Graphs
- Maintaining dictionary
- Gaming
- Evaluation of Arithmetic Expression

THANK YOU

Department of Computer Science and Engineering

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

Problem Solving With C - UE24CS151B

Queue, Priority Queue

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

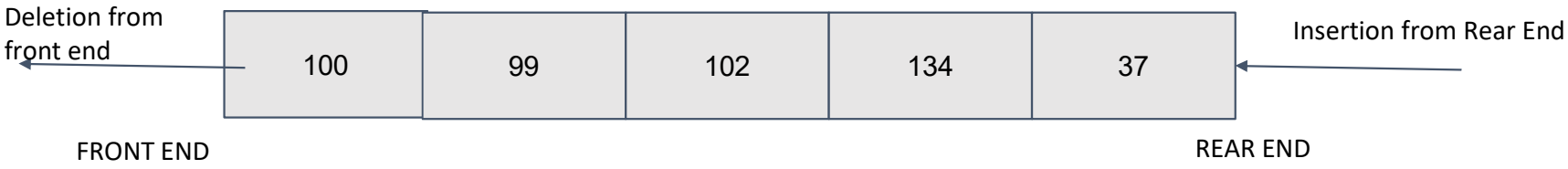
PROBLEM SOLVING WITH C

Queue, Priority Queue

1. Introduction to Queue
2. Operations
3. Types of Queues
4. Introduction to Priority Queue
5. Applications of Priority Queue
6. Implementation methods

Introduction to Queue

- A line or a sequence of people or vehicles awaiting for their turn to be attended or to proceed.
- In computer Science, a list of data items, commands, etc., stored so as to be retrievable in a definite order
- A Data structure which has 2 ends – **Rear end and a Front end**. Open ended at both ends
- Data elements are inserted into the queue from the Rear end and deleted from the front end.
- Follows the Principle of **First In First Out (FIFO)**



Operations on Queue

- Enqueue – Add (store) an item to the queue from the Rear end.
- Dequeue – Remove (access) an item from the queue from the Front end.

Types of Queues

- **Ordinary Queue** - Insertion takes place at the Rear end and deletion takes place at the Front end
- **Priority Queue** - Special type of queue in which each element is associated with a priority and is served according to its priority. If elements with the same priority occur, they are served according to their order in the queue
- **Circular Queue** - Last element points to the first element of queue making circular link.
- **Double ended Queue** - Insertion and Removal of elements can be performed from both front and rear ends

Priority Queue

Introduction

- Type of Queue where each element has a "**Priority**" associated with it.
- Priority decides about the Deque operation.
- The Enque operation stores the item and the “Priority” information
- Types of Priority Queue:

  Ascending Priority Queue: Smallest Number - Highest Priority

  Descending Priority Queue: Highest Number - Highest Priority

PES UNIVERSITY

Applications of Priority Queue

1. Implementation of Heap Data structure.
2. Dijkstra's Shortest Path Algorithm
3. Prim's Algorithm
4. Data Compression
5. OS - Load Balance Algorithm.
6. …

Implementation of Priority Queue

- Using an Unordered Array
- Using an Ordered Array
- Using an Unordered Linked list
- Using an Ordered Linked List
- Using Heap

PES UNIVERSITY

Implementation of Priority Queue Continued..

- Using an Unordered Linked list

  Involves defining three new structures.

  component: Type which contains **details** and **priority**

  ```
  struct component { char details[20]; int priority; };
  ```

  node: Type that contains **component** and a **pointer to itself**

  ```
  struct node{ struct component c; struct node *link; };
  ```

  priority_queue: Type that contains **head which is a pointer to a Node**

  ```
  struct priority_queue { struct node *head; };
  ```

Implementation of Priority Queue Continued..

- Functions involved in the implementation

  Enqueue: This function creates a node using the component from the client and adds this node in the beginning of the queue.

  Dequeue: Deletes a node based on Priority field

  Display: Display the nodes, in turn the components of the priority queue

THANK YOU

Department of Computer Science and Engineering

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