Parul® University

Unit VI – Pointers, Dynamic memory allocation and File Management in C

PROF. MR. HOJIWALA ROBIN

Artificial Intelligence & Data Science

Parul®
University

DIGITAL LEARNING CONTENT

POINTER

- This section introduces the concept of pointer arithmetic, and this will form one of the very important building blocks in understanding the functionality of pointers.

Pointer Expressions and Pointer

1. Arithmetic operations can be performed on pointers
2. Increment/decrement pointer (++ or --)
3. Add an integer to a pointer(+ or += , - or -=)
4. Pointers may be subtracted from each other
5. All these operations meaningless unless performed on an array

6. NOTE: Division and Multiplication are not allowed.

Pointer Expressions and Pointer

- Arithmetic operations can be performed on pointers
  - Increment/decrement pointer (++ or --)
  - Add an integer to a pointer(+ or += , - or -=)
  - Pointers may be subtracted from each other
  - All these operations meaningless unless performed on an array

  - NOTE: Division and Multiplication are not allowed.

Pointer Arithmetic

- *int a,b,*p,*q*
- *p=-q/* illegal use of pointers*/*
- *p<<=1 /* illegal use of pointers*/*
- *p=p-b/*valid*/*
- *p=p-q /* nonportable pointer conversion*/*
- *p=(int*) p-q/*valid*/*
- *p=p-q-a/*valid*/*
- *p=p+a /*valid*/*
- *p=p+q/* invalid pointer addition*/*
- *p=p*q/* illegal use of pointers*/*
- *p=p/q/* illegal use of pointers*/*
- *p=p/a/* illegal use of pointers*/*

Pointer Increment – Example - 1

```
#include<stdio.h>
void main()
{
    int n;
    int   *pn;
    pn=&n;
    int *pn1;
    pn1=pn+1;
    printf("%d  %d\n", pn,pn1);

    int        *pn;
    pn=&n;   int
    *pn1;
    pn1=pn+1;
    printf("%d  %d\n", pn,pn1);
    }
```

2686788 2686792
2686768 2686776

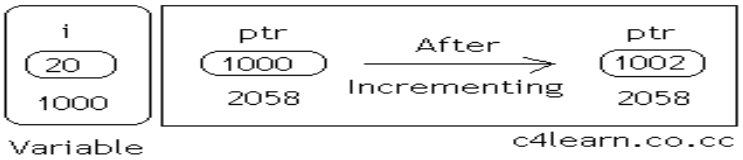
Incrementing Pointer

- Incrementing Pointer is generally used in array because we have contiguous memory in array and we know the contents of next memory location.
- Incrementing Pointer Variable Depends Upon data type of the Pointer variable

Three Rules should be used to increment pointer –

```
Address + 1 = Address

Address++   = Address

++Address   = Address
```

Pictorial Representation :



Data Type	Older Address stored in pointer	Next Address stored in pointer after incrementing (ptr++)
int	1000	1002
float	1000	1004
char	1000	1001

Example – 2

```
#include<stdio.h>
int main()
{
    int *ptr=(int *)1000;
   printf("Old Value of ptr : %u",ptr);
   ptr=ptr+1;
   printf("New Value of ptr : %u",ptr);
   return 0;
}
```

Old Value of ptr : 1000
New Value of ptr : 1004

Difference between two integer Pointers – Example - 3

```
#include<stdio.h>

int main(){

float *ptr1=(float *)1000;
float *ptr2=(float *)2000;

printf("\nDifference : %d\n",ptr2-ptr1);
return 0;
}
```

Difference : 250

Explanation

- Ptr1 and Ptr2 are two pointers which holds memory address of Float Variable.
- Ptr2-Ptr1 will gives us number of floating point numbers that can be stored.
- ptr2 - ptr1 = (2000 - 1000) / sizeof(float)

  = 1000 / 4

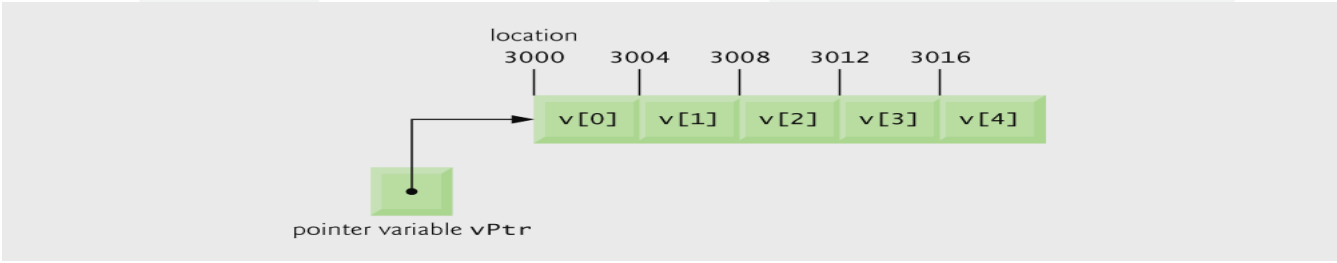
Pointer Division – Example - 4

```
#include<stdio.h>
int main()
{
int *ptr1,*ptr2;
ptr1 = (int *)1000;
ptr2 = ptr1/4;
return(0);
}
```

Illegal Use of operator : INVALID

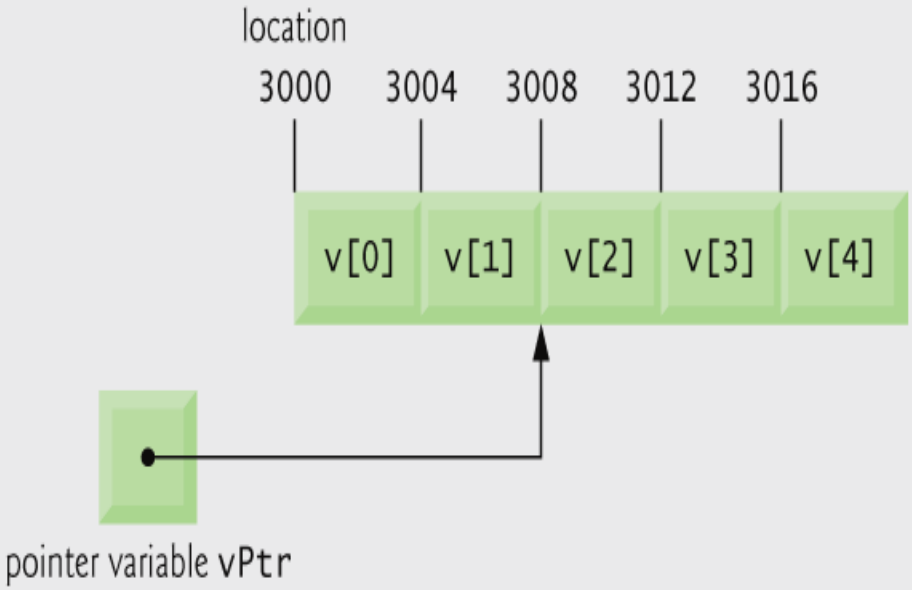
Pointer Expressions and Pointer Arithmetic-Arrays

- 5 element `int` array on machine with 4 byte `int`s
  - `vPtr` points to first element `v[ 0 ]`
    - at location `3000` (`vPtr = 3000`)
  - `vPtr += 2;` sets `vPtr` to `3008`
    - `vPtr` points to `v[ 2 ]` (incremented by 2), but the machine has 4 byte `int`s, so it points to address `3008`



Array **`v`** and a pointer variable **`vPtr`** that points to **`v`**.

The pointer vPtr after pointer



Pointer Expressions and Pointer

- Subtracting pointers
  - *Returns number of elements from one to the other*. If

```
vPtr2 = v[ 2 ];
vPtr = v[ 0 ];
```

  - `vPtr2 - vPtr` would produce 2
- Pointer comparison (<, == , >)
  - See which pointer points to the higher numbered array element
  - Also, see if a pointer points to **0**

The Relationship Between Pointers and Arrays

- Arrays and pointers closely related
  - Array name like a constant pointer
  - Pointers can do array subscripting operations
- Define an array `b[ 5 ]` and a pointer `bPtr`
  - To set them equal to one another use:

```
bPtr = b;
```

  - The array name (`b`) is actually the address of first element of the array `b[ 5 ]`

```
bPtr = &b[ 0 ]
```

  - Explicitly assigns `bPtr` to address of first element of `b`

The Relationship Between Pointers and Arrays

- Element `b[ 3 ]`
  - Can be accessed by `*( bPtr + 3 )`
    - Where `n` is the offset. Called pointer/offset notation
  - Can be accessed by `bptr[ 3 ]`
    - Called pointer/subscript notation
    - `bPtr[ 3 ]` same as `b[ 3 ]`
  - Can be accessed by performing pointer arithmetic on the array itself

    `*( b + 3 )`

Jump Statements

- **return -** The return in C returns the flow of the execution to the function from where it is called. This statement does not mandatorily need any conditional statements. As soon as the statement is executed, the flow of the program stops immediately and returns the control from where it was called.
- The return statement may or may not return anything for a void function, but for a non-void function, a return value must be returned.

Statement –

```
int sum(int a , int b){
        int s = a + b ;
        return s;
}
```

ARRAY OF POINTERS

- An array of pointer is similar to an array of ant predefined data type
- As a pointer variable always contain an address, an array of pointer is a collection of addresses.
- These can be address of ordinary isolated variable or of array elements.
- The elements of an array of pointers are stored in the memory just like elements of any other kind of array.
- Example is given below..

Example - 1

```
#include <stdio.h>
 const int MAX = 3;
 int main () {
   int var[] = {10, 100, 200};
   int i, *ptr[MAX]; //array of pointers
    for ( i = 0; i < MAX; i++) {
      ptr[i] = &var[i]; /* assign the address of integer. */
   }
      for ( i = 0; i < MAX; i++) {
      printf("Value of var[%d] = %d\n", i, *ptr[i] );
   }
   return 0;
 }
```

Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200

Example - 2

```
#include<stdio.h>
void main()
{
   int arr[3]={1,2,3};
   int i, *ptr[3];
   for(i=0;i<3;i++)
         ptr[i]=arr+i;
   for(i=0;i<3;i++)
         printf("%p %d\n", ptr[i],*ptr[i]);
}
```

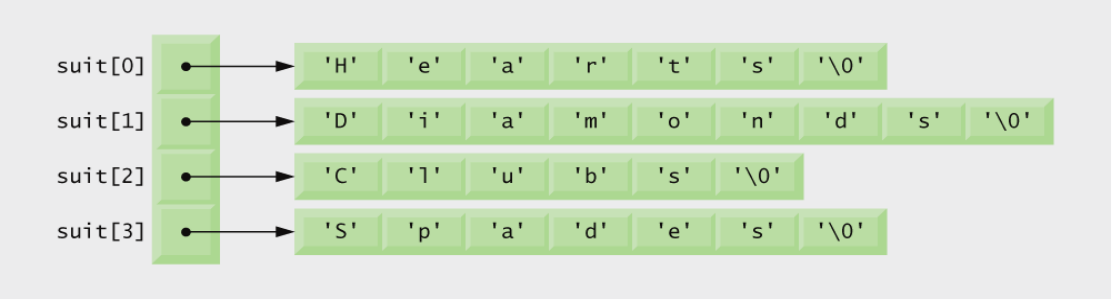
```
0028FF30 1
0028FF34 2
0028FF38 3
```

Arrays of Pointers - Strings

- Arrays can contain pointers
- For example: an array of strings

```
char *suit[ 4 ] = { "Hearts", "Diamonds",
   "Clubs", "Spades" };
```

  - Strings are pointers to the first character
  - `char *` – each element of `suit` is a pointer to a `char`
  - The strings are not actually stored in the array `suit`, only pointers to the strings are stored



  - `suit` array has a fixed size, but strings can be of any size

Case study: Roman numeral equivalents – Example - 3

```
#include <stdio.h>

void main( void ) {
        int decimal_number = 101, a = 0, b = 0;
    const char *x[11] = {"", "x", "xx", "xxx", "xl", "l", "lx", "lxx", "lxxx", "xc",
    "c"};
    const char *y[10] = {"", "i", "ii", "iii", "iv", "v", "vi", "vii", "viii", "ix"};

                while ((decimal_number > 100) | |  (decimal_number < 0)) {
                        printf("Enter the decimal numbers in the range 1 to
                        100:\n"); scanf("%d", &decimal_number);
                }
    a = decimal_number/10;
    b = decimal_number%10;
  printf("The equivalent roman is %s%s\n", x[a],
  y[b]);
                                                                    }
```

Enter the decimal numbers in the range 1 to 100:

15

The equivalent roman is xv

Parul®
University

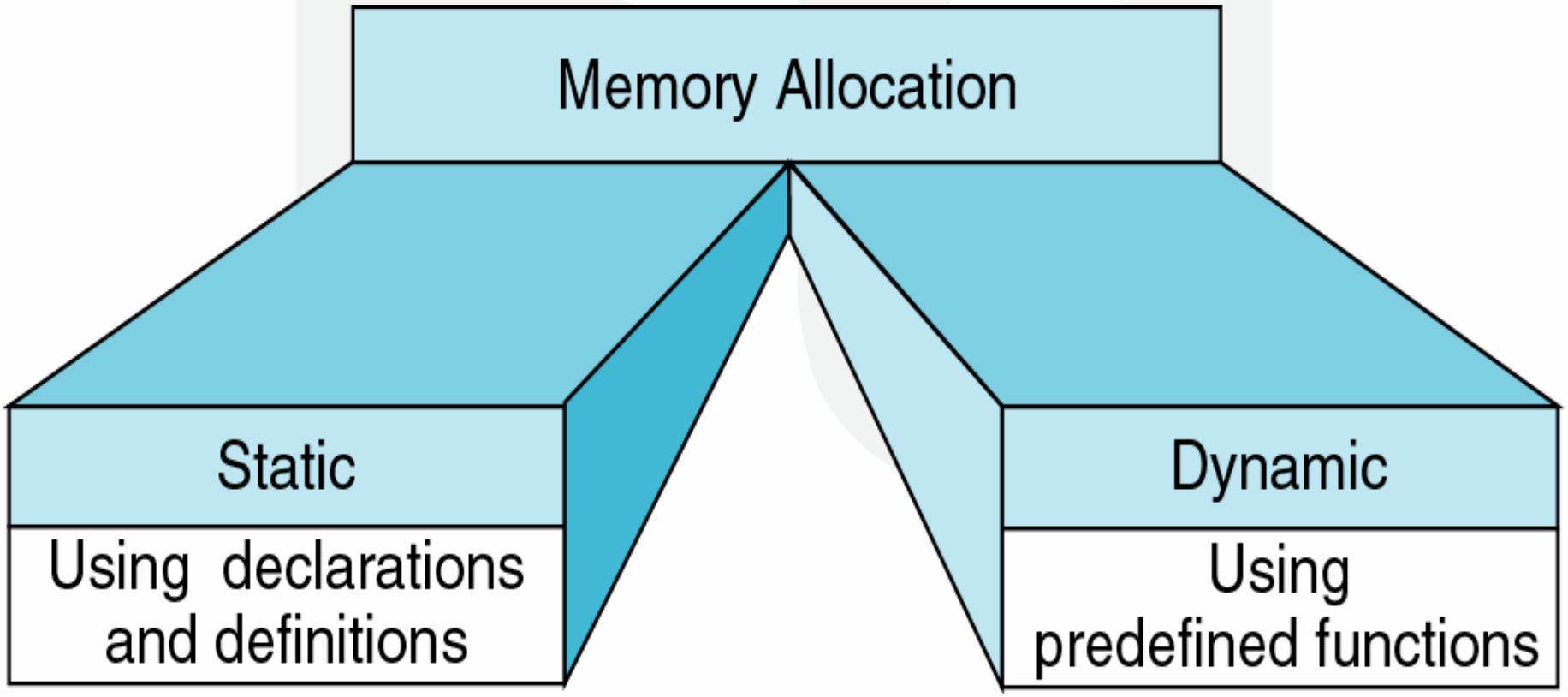
DIGITAL LEARNING CONTENT

DYNAMIC MEMORY ALLOCATION

- Learn how to allocate and free memory, and to control dynamic arrays of any type of data in general and structures in particular.

- Practice and train with dynamic memory in the world of work oriented applications.
- To know about the pointer arithmetic
- How to create and use array of pointers.

Memory Allocation Function



Parul®
University

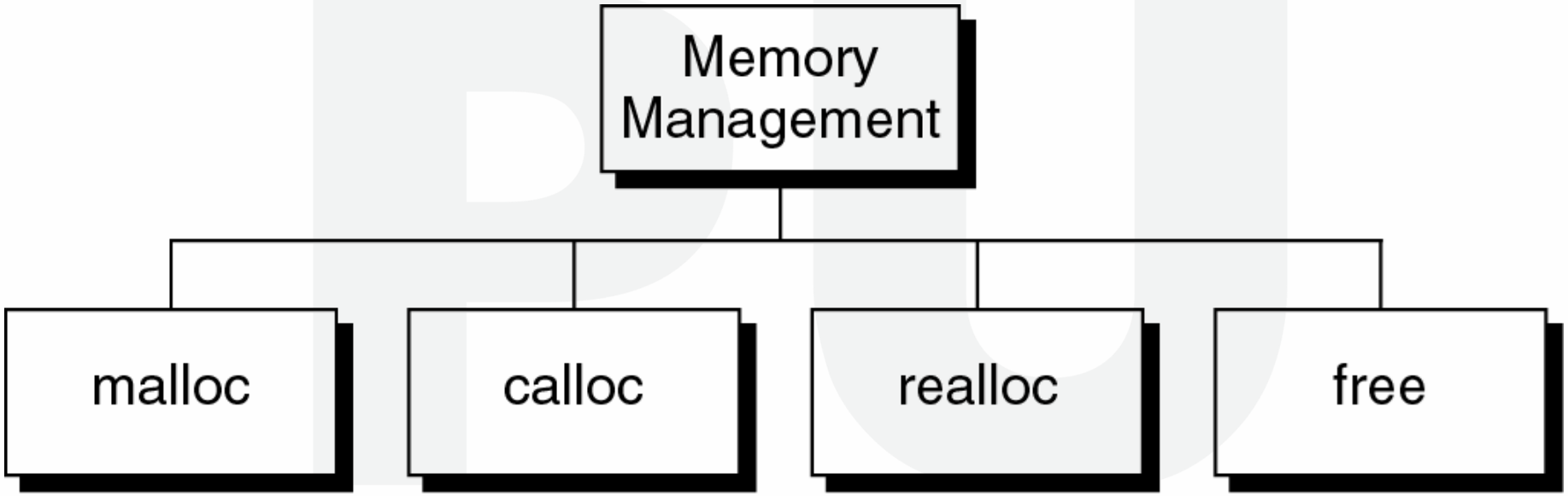
Difference between Static and Dynamic memory allocation

S.no	Static memory allocation	Dynamic memory allocation
1	In static memory allocation, memory is allocated while writing the C program. Actually, user requested memory will be allocated at compile time.	In dynamic memory allocation, memory is allocated while executing the program. That means at run time.
2	Memory size can’t be modified while execution.	Memory size can be modified while execution.
	Example: array	Example: Linked list

Introduction

- Creating and maintaining dynamic structures requires **dynamic memory allocation—** the ability for a program to *obtain more memory space at execution time to hold new values*, and to *release space no longer needed.*

Memory Allocation Functions



Syntax

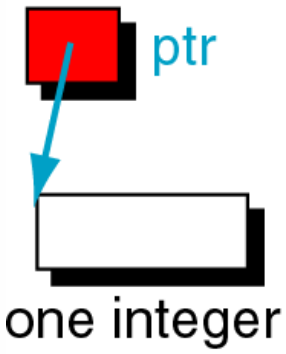
- The following are the function used for dynamic memory allocation
- **void *malloc(int num);**
  - This function allocates an array of **num** bytes and leave them uninitialized.
- **void *calloc(int num, int size);**
  - This function allocates an array of **num** elements each of which size in bytes will be **size**.
- **void *realloc(void *address, int newsize);**
  - This function re-allocates memory extending it upto **newsize**.
- **void free(void *address);**
  - This function releases a block of memory block specified by address.

Block Memory Allocation (malloc)

- Malloc function allocates a block of memory that contains the number of bytes specified in its parameter.
- It returns a void pointer to the first byte of the allocated memory
- The allocated memory is not initialized . We should therefore assume that it will contain unknown values and initialize it as required by our program.
- The function declaration is as follows
  - ***void* malloc (size_t size)***
- If it is not successful malloc return NULL pointer.
- An attempt to allocate memory from heap when memory is insufficient is known as **overflow.**

malloc

- It is up to the program to check the memory overflow
- If it doesn't the program produces invalid results or aborts with an invalid address the first time the pointer is used.




```
if (!(ptr = (int *)malloc(sizeof(int))))
   /* No memory available */
   exit (100) ;
/* Memory available */
...
```

Parul®
University

malloc

- Malloc function has one more potential error.
- If we call malloc function with zero size , the results are unpredictable.
- It may return a NULL pointer
- ***Never call malloc with a zero size!!!!!***

Parul® University

DIGITAL LEARNING CONTENT

Contiguous Memory Allocation (calloc)

- Calloc is primarily used to allocate memory for arrays.
- It differs from malloc only in that it sets memory to null characters.
  - ***void *calloc (size_t element-count, size_t element-size)***


```
if (!(ptr = (int *)calloc (200, sizeof(int))))
    /* No memory available */
    exit (100) ;

/* Memory available */
...
```

Parul®
University

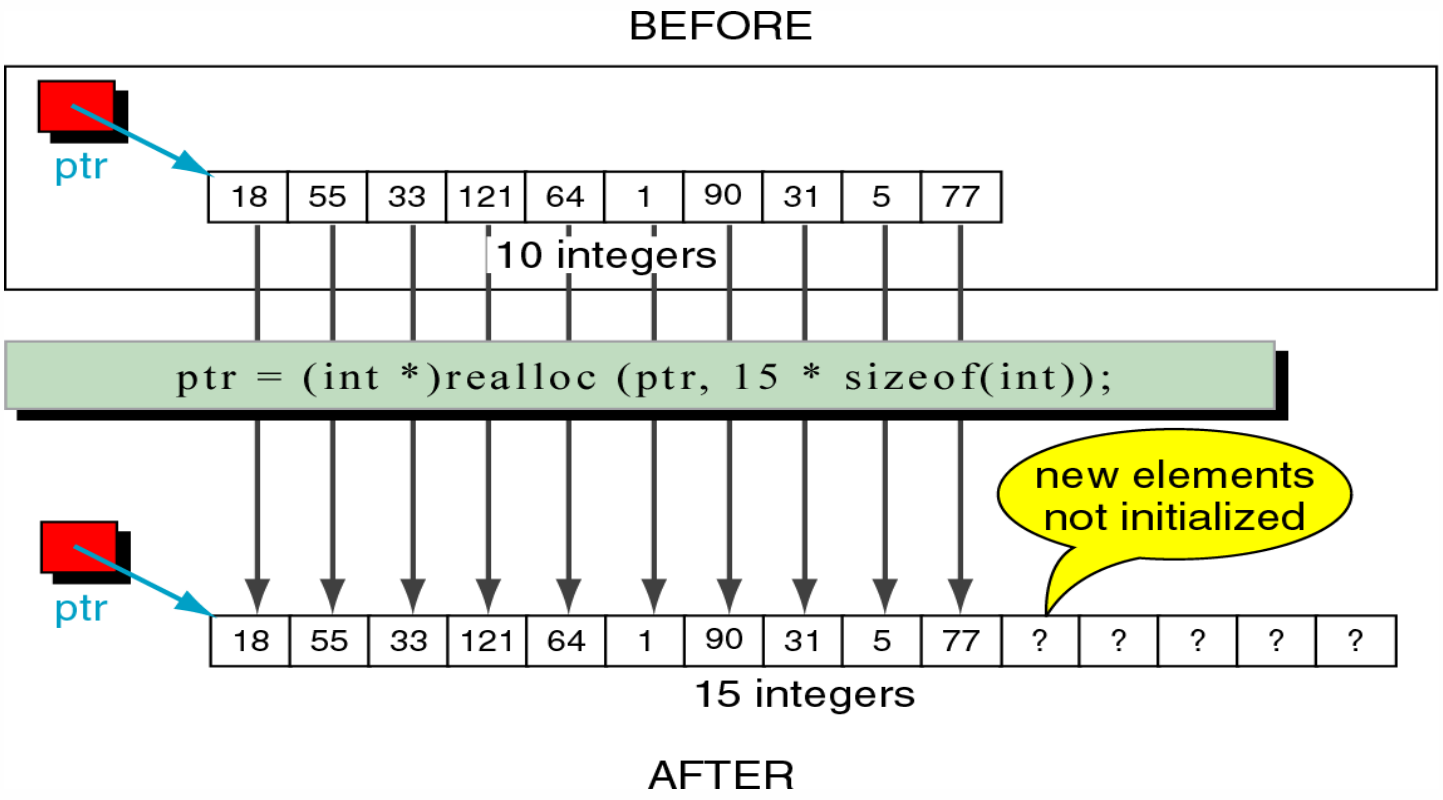
Reallocation of memory(realloc)

- The realloc function can be highly inefficient and should be used advisedly.
- When given a pointer to the previously allocated block of memory, realloc changes the size of the block by deleting or extending the memory at the end of the block.
- If memory cannot be extended because of other allocations, realloc allocates a completely new block and copies the existing memory allocation to new allocation, and deletes the old allocation.
  - ***void *realloc (void* ptr, size_t new/ize)***

Parul®
University

DIGITAL LEARNING CONTENT

realloc



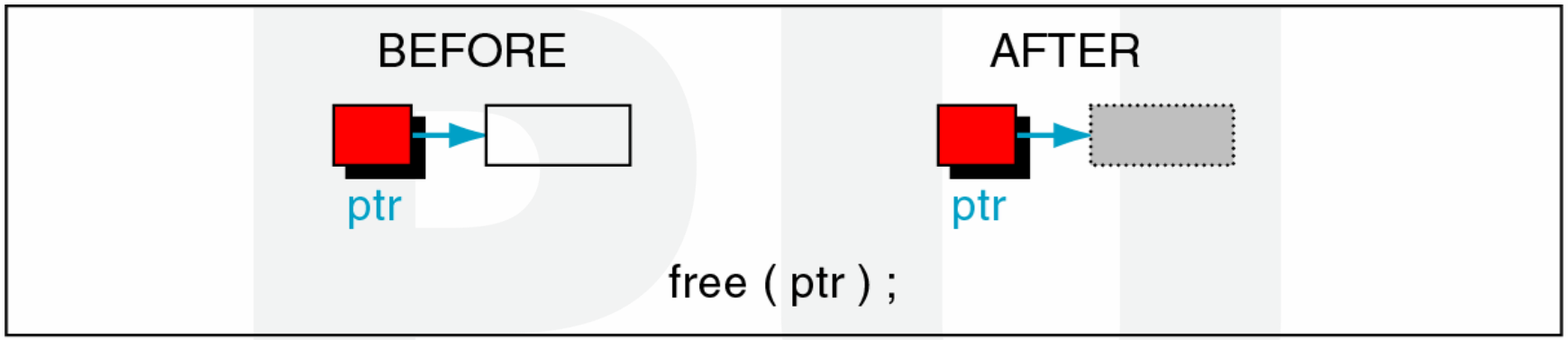
Parul®
University

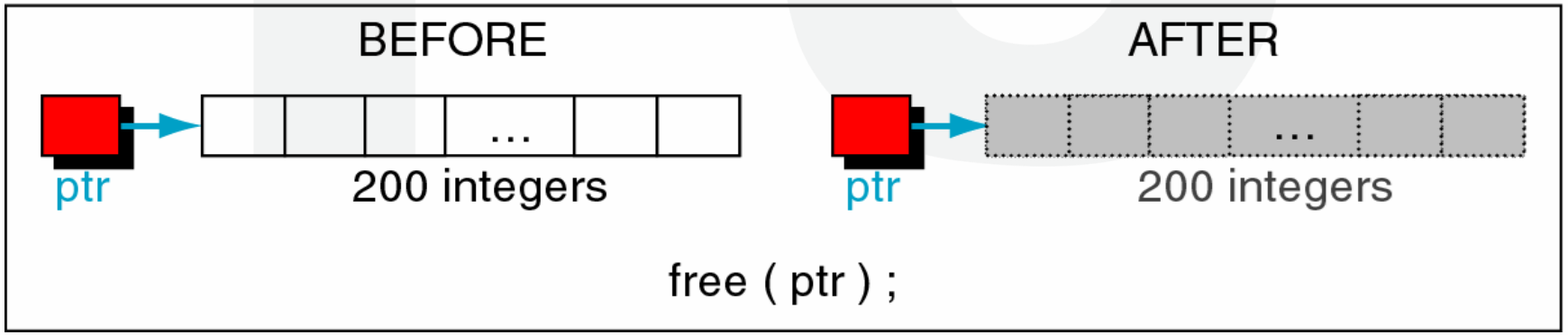
Releasing Memory (free)

- When memory locations allocated by *malloc, calloc or realloc* are no longer needed, they should be freed using the predefined function ***free***.
  - ***void free(void* ptr)***
- Below shows the example where first one releases a single element allocated with malloc
- Second example shows 200 elements were allocated with calloc . When free the pointer 200 elements are returned to the heap.


DIGITAL LEARNING CONTENT

free


BEFORE
AFTER
ptr
ptr
free (ptr) ;


BEFORE
AFTER
ptr
200 integers
ptr
200 integers
free (ptr) ;

Parul® University

Difference between malloc and calloc

S.no	malloc()	calloc()
1	It allocates only single block of requested memory	It allocates multiple blocks of requested memory
2	int *ptr;ptr = malloc(20 * sizeof(int));For the above, 20*4 bytes of memory only allocated in one block.	int *ptr;Ptr = calloc(20, 20 * sizeof(int));For the above, 20 blocks of memory will be created and each contains 20*4 bytes of memory.
	Total = 80 bytes	Total = 1600 bytes
3	malloc () doesn't initializes the allocated memory. It contains garbage values	calloc () initializes the allocated memory to zero
4	type cast must be done since this function returns void pointer int *ptr;ptr = (int*)malloc(sizeof(int)*20);	Same as malloc () function int *ptr;ptr = (int*)calloc(20, 20 * sizeof(int));

Parul®
University

Resizing and Releasing Memory

- When your program comes out, operating system automatically release all the memory allocated by your program but as a good practice when you are not in need of memory anymore then you should release that memory by calling the function **free()**.

- Alternatively, you can increase or decrease the size of an allocated memory block by calling the function **realloc()**.

Parul® University

Example-1

```
#include  <stdio.h>
#include <stdlib.h>
int main(){
   int n,i,*ptr,sum=0;
   printf("Enter number of elements: ");
   scanf("%d",&n);
   ptr=(int*)malloc(n*sizeof(int));  //memory allocated using malloc
   if(ptr==NULL)
   {
      printf("Error! memory not allocated.");
      exit(0);
   }
   printf("Enter elements of array: ");
   for(i=0;i<n;++i)
   {
      scanf("%d",ptr+i);
      sum+=*(ptr+i);
   }
   printf("Sum=%d",sum);
   free(ptr);
   return 0;

        }
```

Enter number of elements: 3
Enter elements of array: 2 7 1
Sum=10

Parul® University

Example - 2

```
#include          <stdio.h>
#include   <stdlib.h>   int
main(){
   int n,i,*ptr,sum=0;
   printf("Enter number of elements: ");
   scanf("%d",&n);
   ptr=(int*)calloc(n,sizeof(int));
   if(ptr==NULL)
   {
      printf("Error! memory not allocated.");
      exit(0);
   }
   printf("Enter elements of array: ");
   for(i=0;i<n;++i)
   {
      scanf("%d",ptr+i);
      sum+=*(ptr+i);
   }
   printf("Sum=%d",sum);
   free(ptr);
   return 0;
}
```

Enter number of elements: 3
Enter elements of array: 2 1 3
Sum=6

Parul® University

DIGITAL LEARNING CONTENT

Example - 3

```
#include <stdio.h> #include
<stdlib.h> int main(){
   int *ptr,i,n1,n2; printf("Enter size of
   array: "); scanf("%d",&n1);
   ptr=(int*)malloc(n1*sizeof(int)); printf("Address of
   previously allocated memory: "); for(i=0;i<n1;++i)

    printf("%u\t",ptr+i); printf("\nEnter
new size of array: "); scanf("%d",&n2);
ptr=realloc(ptr,n2); for(i=0;i<n2;++i)
    printf("%u\t",ptr+i);
return 0;
}
```

```
Enter size of array: 3
Address of previously allocated memory: 7474944
7474948 7474952
Enter new size of array: 5
7474944 7474948 7474952 7474956 7474960
```

Parul® University

Memory Leaks

- A memory leak occurs when allocated memory is never used again but is not freed.
- I can happen when
  - The memory's address is lost
  - The free function is never invoked though it should be
- The problem with memory leak is that the memory cannot be reclaimed and use later. The amount of memory available t the heap manager will be decreased.
- If the memory is repeatedly allocated and then lost, then the program may terminate when more memory is needed but malloc cannot allocate it because it ran out of memory.

Example

```
char *chunk;
while(10
{
        chunk=(char*) malloc (1000000);
        printf("allocating\n");
        }
```

- The variable chunk is assigned memory from heap. However this memory is not freed before another block of memory is assigned to it.
- Eventfully the application will run out of memory and terminate abnormally.

Parul® University

What is a File?

- A named collection of data, typically stored in a secondary storage (e.g., hard disk).

  Examples
  - Records of all employees in an organization
  - Document files created using Microsoft Word
  - Video of a movie
  - Audio of a music

- Non-volatile data storage
  - Can be used when power to computer is off

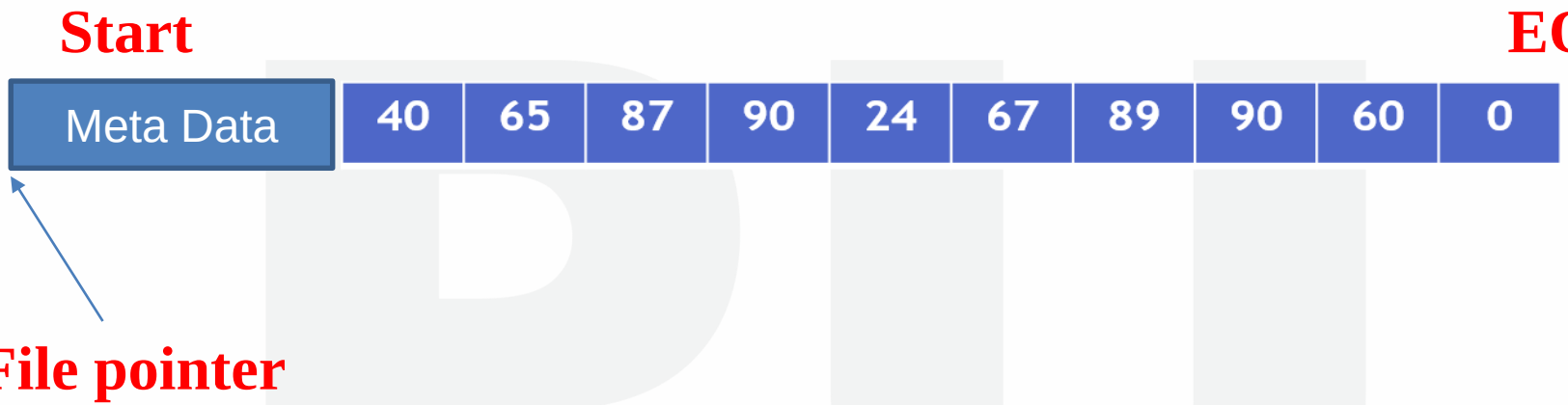
Parul®
University

How a File is Stored?

- Stored as sequence of bytes, logically contiguous (may not be physically contiguous on disk).
  - Discrete storage unit for data in the form of a stream of bytes.
  - Every file is characterized with a starting of file, sequence of bytes (actual data), and end of stream (or end of file).
  - Allow only sequential access of data by a pointer performing.
  - Meta-data (information about the file) before the stream of actual data can be maintained to have a knowledge about the data stored in it.

Parul® University

How a File is Stored?



- The last byte of a file contains the end end-of-file character (EOF, with ASCII code 1A (Hex).
- While reading a file, the EOF character can be checked to know the end.

Parul®
University

Type of Files

- Text files
  - Contain ASCII code only
    - C-programs
- Binary files
  - Contain non-ASCII characters
    - Image, audio, video, executable, etc.

Parul® University

Operations on Files

- Typical operations on a file are
  - Open : To open a file to store/retrieve data in it
  - Read : The file is used as an input
  - Write : The file is used as output
  - Close : Preserve the file for a later use
  - Access: Random accessing data in a file

Parul® University

File Handling Commands

- Include header file <stdio.h> to access all file handling utilities.
- A data type namely FILE is there to create a pointer to a file.

Syntax

```
FILE * fptr;          // fptr is a pointer to file
```

- To open a file, use fopen() function

Syntax

```
FILE * fopen(char *filename, char *mode)
```

- To close a file, use fclose() function

Syntax

```
int fclose(FILE *fptr);
```

Parul® University

fopen() function

- The first argument is a string to characters indicating the name of the file to be opened.
  - The convention of file name should follow the convention of giving file name in the operating system.

Examples:

```
xyz12.c      student.data      File PDS.txt

myFile
```

Parul® University

fopen() function

- The second argument is to specify the mode of file opening. There are five file opening modes in C
  - "r" : Opens a file for reading
  - "w" : Creates a file for writing (overwrite, if it contains data)
  - "a" : Opens a file for appending - writing on the end of the file
  - “rb” : Read a binary file (read as bytes)
  - “wb” : Write into a binary file (overwrite, if it contains data)
- It returns the special value NULL to indicate that it couldn't open the file

Example: Opening a File

```
FILE *fptr;
fptr = fopen("data.txt", "w");
if (fptr == NULL) {
    printf("Error opening file!\n");
    exit(1);
}
```

Parul®
University

File Pointers

Declaration: FILE *fptr;
Purpose: To point to the location of a file in memory.
Used to perform file operations.

Closing a File - fclose()

Syntax: fclose(fptr);
Importance: Releases system resources and ensures data is written to the file.

Writing to a File - fprintf()

Syntax: fprintf(fptr, "format string", variables);
Similar to printf(), but writes to a file.

Parul®
University

Example: Writing to a File

```
fprintf(fptr, "Name: John Doe, Age: 30\n");
```

Reading from a File - fscanf()

Syntax: fscanf(fptr, "format string", &variables);
Similar to scanf(), but reads from a file.

Example: Reading from a File

```
char name[50];
int age;
fscanf(fptr, "Name: %s, Age: %d", name, &age);
```

Example

```
#include <stdio.h>

int main() {
  FILE *file;
  char data[] = "Hello, File Handling in C!";

  // Write to a file
  file = fopen("my_file.txt", "w");
  if (file == NULL) {
    printf("Error opening file for writing.\n");
    return 1;
  }
  fprintf(file, "%s", data);
  fclose(file);

  // Read from a file
  file = fopen("my_file.txt", "r");
  if (file == NULL) {
    printf("Error opening file for reading.\n");
    return 1;
  }
  char buffer[100];
  fscanf(file, "%s", buffer);
  printf("Data read from file: %s\n", buffer);
  fclose(file);

  return 0;
}
```

Reading Character by Character - fgetc()

Syntax: char ch = fgetc(fptr);
Reads a single character from the file.
Returns EOF (End Of File) when the end of the file is reached.

Example

```
#include <stdio.h>

int main() {
    FILE *file;
    char ch;

    // Open file in read mode
    file = fopen("example.txt", "r");

    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    // Read characters until EOF is reached
    while ((ch = fgetc(file)) != EOF) {
        putchar(ch); // Output character to console
    }

    fclose(file);
    return 0;
}
```

Parul® University

Writing Character by Character - fputc()

Syntax: fputc(ch, fptr);
Writes a single character to the file.

Example

```
#include <stdio.h>

int main() {
    FILE *file;
    char text[] = "Hello, World!";
    int i = 0;

    // Open file in write mode
    file = fopen("example.txt", "w");

    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    // Write characters one by one
    while (text[i] != '\0') {
        fputc(text[i], file);
        i++;
    }

    fclose(file);
    printf("Data written to file successfully.\n");
    return 0;
}
```

Parul®
University

Reading Strings - fgets()

Syntax: fgets(str, size, fptr);
Reads a line of characters from the file into the string str.

Example

```
#include <stdio.h>

int main() {
    FILE *file;
    char buffer[100];

    // Open file in read mode
    file = fopen("example.txt", "r");

    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    // Read and display the content line by line
    while (fgets(buffer, sizeof(buffer), file) != NULL) {
        printf("%s", buffer);
    }

    fclose(file);
    return 0;
}
```

Parul®
University

DIGITAL LEARNING CONTENT

Writing Strings - fputs()

Syntax: fputs(str, fptr);
Writes a string to the file.

Example

```
#include <stdio.h>

int main() {
    FILE *file;
    char text[] = "Hello, World!\nThis is fputs().";

    // Open file in write mode
    file = fopen("example.txt", "w");

    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    // Write string to file
    fputs(text, file);

    fclose(file);
    printf("Data written to file successfully.\n");
    return 0;
}
```

Parul® University

Appending to a File

Open the file in append mode ("a").
Use fprintf(), fputs(), or fputc() to add data to the end of the file.

Example

```
#include <stdio.h>

int main() {
    FILE *file;
    char text[] = "Appending new content.\n";

    // Open file in append mode
    file = fopen("example.txt", "a");

    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    // Append string to file
    fputs(text, file);

    fclose(file);
    printf("Data appended to file successfully.\n");
    return 0;
}
```

Results
If the file originally contained:
Hello, World!
This is fputs().
After appending, the file will contain:
Hello, World!
This is fputs().
Appending new content.

Parul®
University

Sequential Access

Data is accessed in a linear, sequential order.
Example: Reading a text file from beginning to end.
Most efficient for reading or writing all data in a file.

Example

```
#include <stdio.h>
int main() {
    FILE *file;
    char text[] = "This is a sequential write
example.\n";
    // Open file in write mode
    file = fopen("example.txt", "w");
    if (file == NUL) {
        printf("Error opening file.\n");
        return 1;
    }
    // Write string to file
    fputs(text, file);
    fclose(file);
    printf("Data written to file successfully.\n");
    return 0;
}
```

Output Example:

✅ Input File (`example.txt`):

```kotlin
Hello, this is an example file.
Sequential access is straightforward.
```

✅ Output:

```kotlin
File contents:
Hello, this is an example file.
Sequential access is straightforward.
```

Parul®
University

Random Access

Data can be accessed in any order.
Uses file positioning functions.
Example: Accessing a specific record in a database file.

Example

```
#include <stdio.h>
int main() {
    FILE *file;
    char text[] = "This is a sequential write
example.\n";
    // Open file in write mode
    file = fopen("example.txt", "w");

    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    // Write string to file
    fputs(text, file);
    fclose(file);
    printf("Data written to file successfully.\n");
    return 0;
}
```

Output Example:

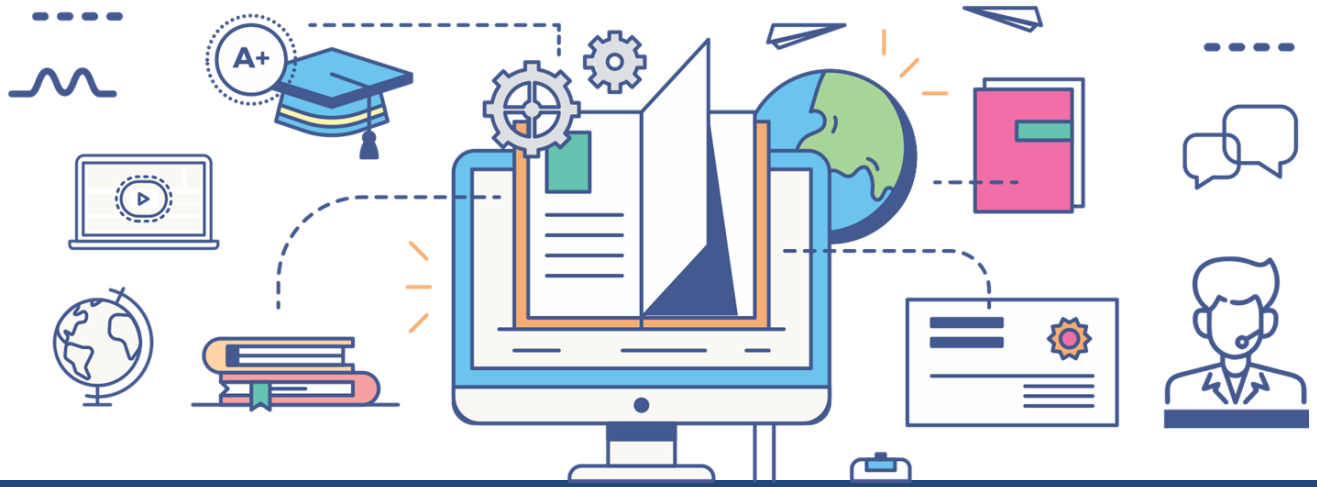
✅ After writing:

```
mathematica

Hello, C Programmers!
```

✅ After reading:

```
css

Data from 7th position: C Programmers!
```

DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in