PADMAPIYUSH'S

LET'S LEARN C

HERE I AM PROVIDING AN EASY CHEATSHEET TO LEARN AND RECALL BEFORE YOUR EXAMS

BASICS

- C is a procedural language that uses functions to perform tasks.
- A C program starts executing from the main() function.
- C is a case-sensitive language.
- C uses semicolons ; to terminate statements.
- Comments can be added using // for a single-line comment, or /* */ for a multi-line comment.

```
#include <stdio.h>

int main() {
  printf("Hello, World!\n");
  return 0;
}
```

DATA TYPES

- **int:** Integer (whole number) values
- **float:** Floating-point numbers (decimal values)
- **double:** Double-precision floating-point numbers
- **char:** Single characters or small strings
- **void:** No return type

OPERTATORS

- +: Addition
- -: Subtraction
- *: Multiplication
- /: Division
- %: Modulus (remainder)
- ++: Increment
- --: Decrement
- =: Assignment

CONDITIONS

- if statement:

```
if(condition){
        //code
    }
```

- if-else statement:

```
if(condition){
    }
else{
    }
```

- Ternary operator:

```
condition ? value_if_true : value_if_false.
```

- if-else-if statement:

```
if(condition){
    }
else if(condition){
    }
else{
    }
```

- Switch Case:

```
switch (expression) {
case value1: // Code to be executed if expression
             is equal to value1
break;
```

LOOPS

- for loop:

```
for(initialization; condition; increment/decrement){
    }
```

- while loop:

```
while(condition){
    }
```

- do-while loop:

```
do{
    }
while(condition);
```

- **break:** Exit loop or switch statement
- **continue:** Skip iteration in loop
- **return:** Return value from function

FUNCTIONS

- A function is a block of code that performs a specific task.
- A function can have multiple parameters, which can be of any data type.
- A function can return a value using the return statement.
- A function can also be called recursively, i.e., a function can call itself.

```
int add(int x, int y);  // Function declaration
int add(int x, int y) {  // Function definition
  int result = x + y;
  return result;
}
int sum = add(3, 5);  // Function call
printf("The sum is %d\n", sum);
```

ARRAYS

- An array is a collection of elements of the same data type.
- Array elements can be accessed using their index value, starting from `0`.
- Array elements can be assigned values using the assignment operator `=`.
- Arrays can be passed as parameters to functions.

```
int arr[5]; // Array declaration
int arr[5] = {1, 2, 3, 4, 5}; // Array initialization
int x = arr[0]; // Accessing array elements
arr[1] = 10;
for (int i = 0; i < 5; i++) { // Array traversal
  printf("%d ", arr[i]);
}
```

POINTERS

- A pointer is a variable that stores the memory address of another variable.
- The & operator is used to obtain the memory address of a variable.
- The * operator is used to dereference a pointer, i.e., to obtain the value stored at the memory address pointed to by the pointer.
- Pointers can be passed as parameters to functions.

```
int *ptr; // Pointer declaration

int x = 10; // Pointer initialization
int *ptr = &x;

int y = *ptr; // Accessing pointer value

*ptr = 20; // Changing pointer value

int *ptr = NULL; // Null pointer
```

STRINGS

- A string is a collection of characters.
- A string is terminated by a null character \0.
- String elements can be accessed using their index value, starting from 0.
- String elements can be assigned values using the assignment operator =.
- Strings can be passed as parameters to functions.

```
char str[10]; // String declaration
char str[10] = "hello"; // String initialization
printf("Enter a string: "); // String input/output
scanf("%s", str);
printf("The string is %s\n", str);

int len = strlen(str); // String length
int cmp = strcmp(str1, str2); // String comparison
```

STRUCTURE

- A structure is a collection of variables of different data types, grouped together under a single name.
- Structure elements can be accessed using the dot ``.`` operator.
- Structures can be passed as parameters to functions.

```
// Structure definition
struct Person {
   char name[50];
   int age;
};
// Structure variable declaration
struct Person p1;
// Structure initialization
struct Person p1 = {"John", 30};
// Accessing structure members
printf("Age: %d\n", p1.age);
```

UNION

- A union is a special data type that allows storing different data types in the same memory location.
- Union elements can be accessed using the dot . operator, like structures.
- Unions can be passed as parameters to functions.

```
// Union definition
union union_name {
  data_type member1;
  data_type member2;
  // ...
} object_name;
// Accessing union members
object_name.member1 = value;
```

STORAGE CLASS

- C provides several storage class specifiers, including **auto**, **register**, **static**, and **extern**.
- **auto** is the default storage class specifier.
- **register** is used to indicate that a variable should be stored in a register.
- **static** is used to indicate that a variable should retain its value between function calls.
- **extern** is used to indicate that a variable or function is defined in another file.

```
// Auto
auto data_type variable_name;
// Register
register data_type variable_name;
// Static
static data_type variable_name;
// Extern
extern data_type variable_name;
```

DYNAMIC MEMORY ALLOCATION

- It allows you to allocate memory dynamically during runtime, instead of at compile time
- **malloc()** function is used to allocate memory dynamically for a single variable or an array.
- **calloc()** function is similar to malloc() but it also initializes the allocated memory block to zero.
- **realloc()** function is used to resize the previously allocated memory block.
- **free()** function is used to deallocate the memory that was previously allocated using **malloc()**, **calloc()** or **realloc()**

```
// malloc
pointer_name = (data_type *) malloc(size);
// calloc
pointer_name = (data_type *) calloc(size,
sizeof(data_type));
// realloc
pointer_name = (data_type *)
realloc(pointer_name, new_size);
// free
free(pointer_name);
```

FILE HALDLING

- C provides several functions for working with files, including fopen(), fclose(), fread(), fwrite(), fseek(), and ftell().
- **fopen()** is used to open a file, and returns a file pointer.
- **fclose()** is used to close a file.
- **fread()** is used to read data from a file.
- **fwrite()** is used to write data to a file.
- **fseek()** is used to set the file position indicator.
- **ftell()** is used to get the current file position.

```
#include <stdio.h>
int main() {
   FILE *fp;
   char buffer[100];
   fp = fopen("example.txt", "r");
   if (fp == NULL) {
      printf("Unable to open file\n");
      return 1;
   }
   fread(buffer, sizeof(char), 100, fp);
   printf("Contents of file: %s\n", buffer);
   fclose(fp);
   return 0;
}
```

GRAPHICS.H

- **graphics.h** is a C library that provides functions for drawing graphics on the screen.
- **graphics.h** is not part of the standard C library and may not be available on all platforms.
- To use graphics.h, you must include the library and call the **initgraph()** function to initialize the graphics system.
- **graphics.h** provides several functions for drawing shapes, **including line()**, **rectangle()**, and **circle()**.

```
//program to draw a rectangle
#include <graphics.h>
int main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "");
   rectangle(100, 100, 200, 200);
   getch();
   closegraph();
   return 0;
}
```

1. Factorial (Without Function)

```
#include <stdio.h>

int main()
{
   int n, i, fact = 1;

   printf("Enter a positive integer: ");
   scanf("%d", &n);

   if (n < 0)
   {
      printf("Factorial of negative integers is not
defined.");
   }
   else{
      for (i = 1; i <= n; i++)
      {
         fact *= i;
      }

      printf("Factorial of %d is %d", n, fact);
   }

   return 0;
}
```

2. Factorial (With Function)

```
#include <stdio.h>
int factorial(int n);
int main(){
   int n;
   printf("Enter a positive integer: ");
   scanf("%d", &n);
   if (n < 0)
   {
      printf("Factorial of negative integers is not
defined.");
   }
   else{
      printf("Factorial of %d is %d", n, factorial(n));
   }
     return 0;
}

int factorial(int n){
   if (n == 0){
      return 1;
   }
   else{
      return n * factorial(n - 1);
   }
}
```

3. Fibonacci

```
#include <stdio.h>

int main()
{
   int n, i, t1 = 0, t2 = 1, nextTerm;

   printf("Enter the number of terms: ");
   scanf("%d", &n);

   printf("Fibonacci Series: ");

   for (i = 1; i <= n; i++)
   {
      printf("%d, ", t1);
      nextTerm = t1 + t2;
      t1 = t2;
      t2 = nextTerm;
   }
   return 0;
}
```

4. Triangle pattern

```
#include <stdio.h>

int main() {
   int rows;

   printf("Enter the number of rows: ");
   scanf("%d", &rows);

   for (int i = 1; i <= rows; i++) {
      for (int j = 1; j <= i; j++) {
         printf("*");
      }
      printf("\n");
   }

   return 0;
}
```

5. Pyramid pattern

```
#include <stdio.h>

int main() {
   int rows, k = 0;

   printf("Enter the number of rows: ");
   scanf("%d", &rows);

   for (int i = 1; i <= rows; i++, k = 0) {
      for (int j = 1; j <= rows - i; j++) {
         printf(" ");
      }
      while (k != 2 * i - 1) {
         printf("*");
         k++;
      }
      printf("\n");
   }

   return 0;
}
```

6. Demonstrate all Operators

```
#include <stdio.h>

int main() {
   int a = 10, b = 20, c = 30, result;

   // Arithmetic operators
   result = a + b;
   printf("a + b = %d\n", result);
   result = a - b;
   printf("a - b = %d\n", result);
   result = a * b;
   printf("a * b = %d\n", result);
   result = b / a;
   printf("b / a = %d\n", result);
   result = b % a;
   printf("b %% a = %d\n", result);
   a++; // equivalent to a = a + 1
   printf("a++ = %d\n", a);
   b--; // equivalent to b = b - 1
   printf("b-- = %d\n", b);

   // Relational operators
   result = a == b;
   printf("a == b: %d\n", result);
   result = b != c;
   printf("b != c: %d\n", result);
   result = c > b;
   printf("c > b: %d\n", result);
   result = a < b;
   printf("a < b: %d\n", result);
   result = b >= c;
   printf("b >= c: %d\n", result);
   result = a <= b;
   printf("a <= b: %d\n", result);

   // Logical operators
   result = (a > b) && (b > c);
   printf("(a > b) && (b > c): %d\n",
result);
   result = (a > b) || (b > c);
   printf("(a > b) || (b > c): %d\n",
result);
   result = !(a > b);
   printf("!(a > b): %d\n", result);

   // Bitwise operators
   result = a & b;
   printf("a & b: %d\n", result);
   result = a | b;
   printf("a | b: %d\n", result);
   result = a ^ b;
   printf("a ^ b: %d\n", result);
   result = ~a;
   printf("~a: %d\n", result);
   result = b << 1;
   printf("b << 1: %d\n", result);
   result = b >> 1;
   printf("b >> 1: %d\n", result);

   // Assignment operators
   result = a;
   printf("a = %d\n", result);
   result += b;
   printf("a += b: %d\n", result);
   result -= b;
   printf("a -= b: %d\n", result);
   result *= b;
   printf("a *= b: %d\n", result);
   result /= b;
   printf("a /= b: %d\n", result);
   result %= b;
   printf("a %%= b: %d\n",
esult);
   result <<= 2;
   printf("a <<= 2: %d\n", result);
   result >>= 2;
   printf("a >>= 2: %d\n", result);
   result &= b;
   printf("a &= b: %d\n", result);
   result |= b;
   printf("a |= b: %d\n", result);
   result ^= b;
   printf("a ^= b: %d\n", result);

   // Conditional operator
   result = (a > b
```