**SHORT Q/A**

**Programming in C :**

All C programs should have a function in it known as main.

Execution starts in function main.

C is case-sensitive.

Comments in C start with /\* and end with \*/. Comments can be spanned over many lines.

C is known as a “free format” language.

All C statements should end with a semicolon (;).

The starting #include <stdio.h> statement instructs the C compiler to preprocess(#) and insert or include the entire contents of file stdio.h(standard input out header file) in its place and compile the resulting file.

**Character Set:**

The characters are used to form expressions, words, and numbers, depending upon the type of computer on which the program will run. The characters in C are grouped into the following categories: Letters, Digits, Special Characters, and White spaces.

**C Tokens**:

The smallest individual units are known as C tokens. C has five types of tokens: Keywords, Identifiers, Constants, Operators, and Special symbols.

**Keywords**: All keywords are sequences of characters that have one or more fixed meanings. All C keywords must be written in lowercase letters.

Example: break, char, int, continue, default, do, etc.

**Identifiers**: A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z, a to z, or an underscore ‘\_’ followed by zero or more letters, underscores, and digits (0 to 9).

**Constants**: Fixed values that do not change during the execution of a C program.

Example: 100 is an integer constant, ‘a’ is a character constant, etc.

**Operators**: The operator is a symbol that indicates and instructs the computer to perform mathematical or logical calculations.

**Example**: Arithmetic operators, Conditional operators, Bitwise operators, Logical operators, etc.

**Delimiters / Separators**: These are used to separate variables, statements, and constants.

Example: comma, semicolon, double quotes, apostrophes, blank space, etc.

Strings: String constants are specified in double-quotes.

“Exampgate exameexam” is a string constant as it is inside double-quotes.

**Variable**:

A variable is a name provided to a storage space that will be used by our programs.

The range of values that can be accommodated within that memory and the set of operations can be applied to the variable.

Every variable in C has a specific data type, and it identifies the size and layout of the variable’s memory.

**Data Type**:

A data type in a programming language collects data with its values having predefined characteristics such as integers and characters.

**Types of Operators :**

**Operators. Symbol**

**Arithmetic operators +, -, \*, /, %, ++, --**

**Assignment operator =, +=, -=, \*=, etc**

**Relational operators <, <=, >, >=, !=, ==**

**Logical operators &&, ||, !**

**Bitwise operators &, |, ~, ^, <<, >>**

**Special operators sizeof(), comma, →**

**Pointer operators \* - Value at address (indirection), & - Address Operator**

**Type Conversions**

**Implicit Type Conversion**: There are certain cases in which data will get automatically converted from one type to another:

When data is being stored in a variable, the data being stored does not match the type of the variable.

The data is stored will be converted to match the type of the storage variable.

When an operation is being performed on data of two different types, the “smaller” data type will be converted to match the “larger” type.

The following example converts the value of x to a double-precision value before performing the division. Note that if the 3.0 were changed to a simple 3, then integer division would be performed, losing any fractional values in the result.

Average = x / 3.0;

When data is passed to or returned from functions.

**Explicit Type Conversion**: Data may also be expressly converted using the typecast operator.

The following example converts the value of x to a double-precision value before performing the division. ( y will then be implicitly promoted, following the guidelines listed above. )

Average = ( double ) x / y;

Note that: x itself is unaffected by this conversion.

**Expression**:

Lvalue: Expressions that refer to a memory location are called “lvalue” expressions.

An lvalue may appear as either the left-hand or right-hand side of an assignment.

Variables are lvalues, and so they may appear on the left-hand side of an assignment.

Rvalue: The term rvalue refers to a data value that is stored at some address in memory.

An rvalue is an expression that cannot have a value assigned to it, which means an rvalue may appear on the right-hand side but not on the left-hand side of an assignment.

Numeric literals are rvalues and so they may not be assigned and cannot appear on the left-hand side.

**C Flow Control Statements :**

A control statement is one of the instructions, statements, or groups of statements in a programming language that determines the execution of other statements or instructions. C provides two types of flow controls.

Looping (decides how many times to perform a certain action)

Branching (deciding what action to take)

**If Statement**:

It takes an expression inside the parenthesis and a statement in it. Expressions are assumed to be true if evaluated values are non-zero. Syntax of if Statement:

(i)

If (condition) statement

(ii)

If (condition){statement1statement2:}

**If Else Statement:**

If (condition){ statements }else{ statements }

**The switch Statement:**

The switch statement tests the given variable’s value (or expression) against a list of few case values. If a case is matched, the block of statements associated with that case is executed:

Switch (control variable)

{

Case constant-1: statement(s);

Break;

Case constant-2: statement(s);

Break;

:

Case constant-n: statement(s);

Break;

Default: statement(s);

}

**The Conditional Operators (? : ):**

The ?: operators are like an if-else statement except that we can use it within expressions because it is an operator. ?: are ternary operators in that it takes three values. They are the only ternary operator in the C language.

Flag = (x < 0) ? 0 : 1;

This conditional statement can be evaluated as following with equivalent if else statement.

If (x < 0) flag = 0; else flag = 1;

**Loop Control Structure :**

Loops are the way to repeat controls and commands. This involves repeating a portion of the program a specified number of times until a particular condition is satisfied.

**While Loop:**

Initialize loop counter;

While (test loop counter using a condition/expression)

{

<Statement1>

<Statement2>

…

< decrement/increment loop counter>

}

**For Loop**:

For (initialize counter; test counter; increment/decrement counter)

{

<Statement1>

<Statement2>

…

}

**Do while Loop:**

Initialize loop counter;

Do

{

<Statement1>

<Statement2>

…

}

While (this condition is true);

**Example:** Print numbers from 1 to 5 using for loop

#include <stdio.h>

int main()

{ int counter;

for(counter=1; counter <= 5; counter++) // loop 5 times

{ printf(“%d “, counter);

}

return 0;

}

**The break Statement:** The break statement is used to instantly jump out of a loop without waiting to get out of the conditional test.

**The continue Statement:** The ‘continue’ statement takes control of the Loop’s beginning, bypassing the statement inside the Loop, which has not yet been executed.

**Goto Statement:** C supports an unconditional control statement, goto, to transfer the control from one point to another in a C program.

C Variable Types A variable is just a named area of storage that can hold a single value. There are two **main variable types:** Local variable and Global variable.

**Local Variable: T**he local variable’s scope is confined within the block or function, where it is defined.

**Global Variable**: Global variable is defined at the top of the program file, and it can be visible and modified by any function that may reference it.

Global variables are initialized automatically by the system when we define them. If the same variable name is used for global and local variables, then the local variable takes preference in its scope. Storage Classes in CA variable name identifies some physical location within the computer, where the string of bits representing the variable’s value is stored.

There are two kinds of locations in a computer where such a value may be kept: Memory and CPU registers.

The variable’s storage class determines in which of the above two types of locations the value should be stored.

**Operator Precedence Relations:** Operator precedence relations in highest to lowest order are as follows:

**Example**: Displaying a character ‘A’ and its ASCII value.

#include <stdio.h>

Int main()

{

Char first\_letter;

First\_letter = ‘A’;

// display character

Printf(“Character %c\n”, first\_letter);

// display ASCII value

Printf(“ASCII value %d\n”, first\_letter);

Return 0;

}

***Functions***

A function is known as a self-contained block of statements that can perform a coherent task of some kind.

A function in C can perform a particular task and supports the concept of modularity.

Making function is indirectly isolating one block of code from other independent code blocks.

**Syntax of Function Definition: r**eturn\_data\_type function\_name (data\_type variable1, data\_type variable2, … )

{

Function\_body

}

A function definition in C programming consists of a function header and a function body.

**Return Type:** A function may return a value. The return\_type is the data type of the value the function returns. Some functions may perform the desired operations without returning a value. In this case, the return\_type is the keyword void.

**Function Name**: This is the actual name of the function. The function name and the parameter list **together** constitute the function signature.

**Parameters**: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as an actual parameter or argument.

**Function Body: T**he function body contains a collection of statements that define what the function does.

Syntax of Function Declaration:

Return\_type function\_name( parameter list );

A function declaration tells the compiler about a function name and how to call the function.

**Example1**: Multiplication of two numbers using function

#include <stdio.h>

Int multiplication(int, int); // function declaration (prototype)

Int multiplication(int x, int y)

{ x = x \* y;

return x; }

// function definition int main()

{ int I, j, k;

scanf(“%d%d”,&I, &j);

k = multiplication(I, j); // function call

printf(“%d\n”, k);

return 0; }

**Example2**: Program to calculate factorial of given number

#include <stdio.h>void factorial( int );

/\* ANSI function prototype

\*/void factorial( int n )

{ int I, factorial\_number = 1;

for( I = 1; I <= n; ++I )

factorial\_number \*= I;

printf(“The factorial of %d is %d\n”, n, factorial\_number );

}

main()

{ int number = 0; printf(“Enter a number\n”);

scanf(“%d”, &number );

factorial( number );

}

**There are 4 types of functions based on return type:**

Function with arguments but no return value

Function with arguments and return value

Function without arguments and no return value

Function without arguments but a return value.

While calling a function in the C program, there are two ways in which arguments can be passed to a function.

**Call by Value**: If we pass values of variables to the function as parameters, such a function call is known as call by value.

**Call by Reference**: Variables are stored somewhere in memory. So, instead of passing the value of a variable, if we pass the location number/address of the variable to the function, it would become

**A call by reference’**.

A function that can call itself such a process is called recursion. Functions can be of two types: (i) Library functions , and (ii) User-defined functions.

**Pointers**

A pointer is a variable that stores a memory address. Like all other variables, it also has a name, has to be declared, and occupies some spaces in memory. It is called a pointer because it points to a particular location.

‘&’ = Address of operator

‘\*’ = Value at address operator or ‘indirection’ operator

&I return the Address of the variable i.

\*(&i) return the value stored at a particular address printing the value of \*(&i) is the same as printing the value of i.

**Pointers are useful due to following reasons:**

They enable us to access a variable that is defined outside a function.

Pointers are more efficient in handling data tables and sometimes even arrays.

Pointers tend to reduce the length and complexity of a program.

They increase the execution speed.

The use of pointers allows easy access to character strings.

***NOTE***:

Int \*p; /\* p is going to store address of integer value (size of memory is 2 bytes) \*/

Float \*q; /\* q is going to store address of floating value (size of memory is 2 bytes) \*/

Char \*ch; /\*ch is going to store address of character variable (size of memory is 2 bytes) \*/

**Pointer Declaration:** data-type \*pointer-name;

**NULL Pointers**: Uninitialized pointers start with random unknown values, just like any other variable type.

Accidentally using a pointer containing a random address is one of the most common errors encountered when using pointers and potentially one of the hardest to diagnose since the errors encountered are generally not repeatable.

**Example**: Program to swap two integer numbers using

#include <stdio.h>void swap(int \*a,int \*b);int main(){ int a,b; a = 5; b = 10; printf(“\nBefore swapping: “);

Printf(“a= %d: b= %d”, a, b);

Swap(&a, &b); // function call

Printf(“\nAfter swapping: “);

Printf(“a= %d: b= %d”, a, b);

Return 0;

}

Void swap(int \*a, int \*b)

{

Int x;

X = \*b; \*b = \*a; \*a = x;

}

**Pointers and Strings:**

A string is an array of characters. Strings in C are handled differently than most other languages. A pointer is used to keep track of a text string stored in memory. It will point to the first character of the string. By knowing the beginning address and the length of the string, the program can locate it.

**Example**: char \*a; a = “Hello World!”;

**Constant pointer:** A pointer is said to be a constant pointer when its pointing cannot be changed.

**Syntax**: <type-of-pointer> \*const <name-of-pointer>

**Example**: int \* const p;

**C Pointer to Constant**: This type of pointer cannot change the value at the Address pointed by it.

**Syntax**: const <type-of-pointer> \*<name-of-pointer>;

**Example**: const int \* p;

**Pointer to Pointer: I**t is a special pointer variable that can store the address of another pointer variable. So it is completely legal for a pointer to be pointing to another pointer.

**Syntax**: <type-of-pointer> \*\* <name-of-pointer>;

**Example**: int \*\*p;

**Array of Pointers:** An array of pointers can be declared as follows.

**Declaration Syntax**: <type> \*<name>[<number-of-elements];

Example: int \*p[3];

**Pointer to an Array:** Pointer to an array can be declared as follows.

**Declaration Syntax:** <type> (\*<name>) [<number-of-elements>];

**Example**: int (\*p)[3];

**Function Pointers:** A function pointer is the same as a pointer to a function.

**Syntax**: <return type of function> (\*<name of pointer>) (type of function arguments);

**Example**: int (\*f)(int, int); // declaration .

Combinations of \* and ++

\*p++ accesses the thing pointed to by p and increments p

(\*p)++ accesses the thing pointed to by p and increments the thing pointed to by p

\*++p increments p first and then accesses the thing pointed to by p

++\*p increments the thing pointed to by p first and then used it in a larger expression.

**Pointer Arithmetic:**

**Assignment**: You can assign an address to a pointer. Typically, you do this by using an array name or using the address operator (&).

**Value finding (dereferencing):** The \* operator gives the value stored in the pointed-to location.

**Taking a pointer address:** Like all variables, pointer variables have an address and a value. The & operator tells you where the pointer itself is stored.

**Adding an integer to a pointer:** You can use the + operator to add an integer to a pointer or a pointer to an integer. In either case, the integer is multiplied by the number of bytes in the pointed-to type, and the result is added to the original Address.

**Incrementing a pointer: I**ncrementing a pointer to an array element makes it move to the next element of the array.

**Subtracting an integer from a pointer:** You can use the – operator to subtract an integer from a pointer; the pointer must be the first operand or a pointer to an integer. The integer is multiplied by the number of bytes in the pointed-to type, and the result is subtracted from the original Address.

Note that there are two forms of subtraction. You can subtract one pointer from another to get an integer, and you can subtract an integer from a pointer and get a pointer.

**Decrementing a pointer**: You can also decrement a pointer. In this example, decrementing ptr2 makes it point to the second array element instead of the third. Note that you can use both the prefix and postfix forms of the increment and decrement operators.

**Subtraction**: You can find the difference of two pointers. Normally, you do this for two pointers to elements that are in the same array to find out how far apart the elements are. The result is in the same units as the type size.

**Comparisons**: You can use the relational operators to compare the values of two pointers, provided the pointers are of the same type.

**Problems with Pointers:**

Assigning Value to Pointer Variable

Assigning Value to Uninitialized Pointer

Not dereferencing Pointer Variable (forgetting to use \* before variable)

Assigning Address of Un-initialized Variable

Comparing Pointers that point to different objects

Dangling pointers (Using free or de-allocating memory or out of**Structur**

**Structure**

***Keyword***

eyword struct is used to define a structure.

***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 member

***Memory***

Each member within a structure is assigned unique storage area of location

***Value Altering***

Altering the values of a member will not affect other members of the structure.

***Accessing Members***

Individual member can be accessed at a time.

***Initialization of Members***

Several member of a structure can initialize at once.