

**Q. 1) Is it possible to execute code even after the program exits the `main()` function?**

**Answer:** The standard C library provides a function named `atexit()` that can be used to perform "cleanup" operations when your program terminates. You can set up a set of functions you want to perform automatically when your program exits by passing function pointers to the `atexit()` function.

**Q. 2) What is the difference between the functions `rand()`, `random()`, `srand()` and `randomize()`?**

**Answer:** `rand()`: `Rand` uses a multiplicative random number generator with period 232 to return successive pseudo-random numbers in the range 0 to `rand-max`.

`Random()`: `Random` returns a random number between 0 and `(num-1)`. `random(num)` is a macro defined in `stdlib.h`.

`Randomize()`: `Randomize` initializes the random number generator with a random value. Because `randomize` is implemented as a macro that calls the time function prototyped in `time.h`, you should include `time.h` when you use this module.

`Srand()`: the random number generator is reinitialized by calling `srand` with an argument value of 1. the generator can be set to a new starting point by calling `srand` with a given seed number.

**Q. 3) Write a Program to convert decimal to binary no.**

**Answer:**

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
int main(int argc, char *argv[])
{
    double num;
    long dec;
    double frac;
    char decstr[1024], *decptr, fracstr[1024], *fracptr, *fracendptr;
    printf("Please enter a decimal number: ");
    scanf("%lf", &num);
    dec = (long)((long)num);
    frac = num - (double)dec;
    bzero(decstr, sizeof(decstr));

    decptr = decstr + sizeof(decstr) - 1;
```

```
while(dec > 0 && decptr > decstr)
{
    decptr--;

    if(dec%2)
        *decptr = '1';
    else
        *decptr = '0';
    dec = dec/2;

}

if(decptr <= decstr)
{
    printf("Oops...Buffer offerflow prevented. Terminating.");
    exit(-1);
}

bzero(fracstr, sizeof(fracstr));
fracptr = fracstr; fracendptr = fracstr + sizeof(fracstr) - 1;

while(frac > 0 && fracptr < fracendptr)
{
    frac = frac*2;
    if((int)(frac))
        *fracptr = '1';
    else
        *fracptr = '0';
    frac = frac - (double)((int)frac);
    fracptr++;
}

if(fracptr >= fracendptr)
{
    printf("Oops...Buffer offerflow prevented. The fractional
    part may not be exactly accurate. But its pretty close.");
}

printf("
Binary: %s.%s", decptr, fracstr);
return 0;
}
```

#### Q. 4) How do we get Square root of any number Without using sqrt() function?

**Answer:** You can use an iterative method for example bisection

```
#include <stdio.h>
main()
{
    int n;
    float eval=1.0,x1=0,x2=n,x;

    //trying to locate root between x1=0 and x2=n and giving a initial dummy value to eval.

    x=(x1+x2)/2;

    printf("Enter value for n n n");
    scanf("%d",&n);

    while((eval > 0.0001 )||( eval < -0.0001) )
    {
        //you can change accuracy by changing 0.0001 to smaller value

        eval= x*x-n;
        printf(" eval=%f t",eval);

        //printf to keep track not necessary
        if(eval>0)
        {
            x2=x;
            x=(x2+x1)/2 ;
            printf(" x = %f t",x);

            //printf to keep track not necessary
        }

        else
        {
            x1=x ;
            x=(x1+x2)/2 ;

            printf(" x = %f t",x);

            //printf to keep track not necessary
        }

        printf("n n");
    }

    printf("the root is = %f ",x);
}
```

**Q. 5) Can you use the function `fprintf()` to display the output on the screen?**

**Answer:** Use it as:

```
fprintf( stdout, "Hello %s", name );
```

this line prints Hello on the console. to print "Hello" in a file, put the file pointer in the place of `stdout`

eg:

```
fprintf( fp, "Hello %s", name );
```

however, the file should be opened in write mode since we are writing.

**Q. 6) How to see return value of main function?**

**Answer:**

//this code will print return value of main

```
static unsigned int nCount = 0;
void NoReturnFunc();
int main()
{
    if(0 == nCount)
        NoReturnFunc();
    return 0;
}
void NoReturnFunc()
{
    nCount++;
    int nRetVal = main();
    printf("Return by main %d", nRetVal);
    getch();
}
```

**Q. 7) What is a static function?**

**Answer:** A static function is a function whose scope is limited to the current source file. Scope refers to the visibility of a function or variable. If the function or variable is visible outside of the current source file, it is said to have global, or external, scope. If the function or variable is not visible outside of the current source file, it is said to have local, or static, scope.

**Q. 8) How to print a statement without using `printf()` in c?**

**Answer:**

using `getchar()`

`main()`

```
{
    int i;
    char c;
    for(i=0;i!=""; i++)
    {
        c=getchar();
        putchar(c,n);
    }
    getch();
}
```

**Q. 9) Have you heard of "mutable" keyword?**

**Answer:** The mutable keyword can only be applied to non-static and non-const data members of a class. If a data member is declared mutable, then it is legal to assign a value to this data member from a const member function.

**Q. 10) Does there exist any other function which can be used to convert an integer or a float to a string ?**

**Answer:** Some implementations provide a nonstandard function called `itoa()`, which converts an integer to string.

```
#include
char *itoa(int value, char *string, int radix);
```

Description: The `itoa()` function constructs a string representation of an integer.

**Parameters**

**Value:** Is the integer to be converted to string representation.

**string:** Points to the buffer that is to hold resulting string. The resulting string may be as long as seventeen bytes.

**radix:** Is the base of the number; must be in the range 2 - 36. A portable solution exists. One can use `sprintf()`:

```
char s[SOME_CONST];
```

```
int i = 10;
float f = 10.20;

sprintf ( s, "%d %f", i, f );
```

**Q. 11) Is using `exit()` the same as using `return`?**

**Answer:** No. The `exit()` function is used to exit your program and return control to the operating system. The return statement is used to return from a function and return control to the calling function. If you issue a return from the `main()` function, you are essentially returning control to the calling function, which is the operating system. In this case, the return statement and `exit()` function are similar.

**Q. 12) What is an argument ? differentiate between formal arguments and actual arguments?**

**Answer:** An argument is an entity used to pass the data from calling function to the called function. Formal arguments are the arguments available in the function definition. They are preceded by their own data types. Actual arguments are available in the function call.

**Q. 13) Differentiate between a linker and linkage?**

**Answer:** A linker converts an object code into an executable code by linking together the necessary build in functions. The form and place of declaration where the variable is declared in a program determine the linkage of variable.

**Q. 14) What is meant by `malloc` function?**

**Answer:** `malloc` function will allocate memory for pointers. Whenever you define a pointer it will just allocate 4 bytes to store the address of pointer and no memory for variable storage. Using `malloc` you can assign memory so that we can store variables and access through pointer for e.g

```
int *p;
*p=5;
```

```
/* It will generate error as no memory allocated for pointer */
```

```
int *p;  
p= malloc (4);
```

```
/* allocates 4 bytes */
```

```
*p=5;
```

```
/* Now it works fine */
```

### Q. 15) How can send unlimited no of arguments to a function, eg printf function can take any no of arguments?

**Answer:** Sending the unlimited number of arguments to a function can be done by "Variable number of arguments" concept in C.

These function definitions will contain an `ellipse(...)` which indicates the variable number of argument. example

```
#include <stdio.h>  
#include <stdarg.h>  
void eprintf (const char *template, ...)  
{  
    va_list ap;  
    extern char *program_invocation_short_name;  
    fprintf (stderr, "%s: ", program_invocation_short_name);  
    va_start (ap, template);  
    vfprintf (stderr, template, ap);  
    va_end (ap);  
}
```

Here `va_start()` initializes variable argument list pointer `ap` to the beginning of the variable argument list, before any calls to `va_arg()`.

The `va_arg()` macro returns the next argument in the variable argument list pointed to by `ap`.

### Q. 16) How would you use the functions `randomize()` and `random()`?

**Answer:** `Random()` returns a random number between 0 and `(num-1)`. `random(num)` is a macro defined in `stdlib.h`.

`Randomize()` initializes the random number generator with a random value. Because `randomize` is implemented as a macro that calls the time function prototyped in `time.h`, you should include `time.h` when you use this routine.

**Q. 17) How to write a C program to find the power of 2 in a normal way and in single step?**

**Answer:** assume : x is input number and want to check whether x is power of two or not.

```
code: result = x & (x-1);
if( result == 1)
x is not power of two
else
x is power of two.
```

**Q. 18) What is a difference between printf and cout and why printf called a function and cout object as both are used to print data?**

**Answer:** In high level terms, the main differences are type safety (`cstdio` doesn't have it), performance (most `iostreams` implementations are slower than the `cstdio` ones) and extensibility (`iostreams` allows custom output targets and seamless output of user defined types)

**Q. 19) What is the purpose of main( ) function?**

**Answer:** `main()` is the user defined function. `main()` is the first function in the program which gets called when the program executes. The startup code contains `runmain()` function which calls `main()` function. we can't change the name of the `main()` function.

**Q. 20) How to convert decimal to octal and hexadecimal in c program?**

**Answer:**

```
main()
{
    int num=12;

    printf("%d in Decimal",num);
    printf("%o in Octal",num);
    printf("%X in Hexa Decimal",num);
}
```

**Q. 21) What is the difference between `printf()` and `sprintf()` ?**

**Answer:** `sprintf()` writes data to the character array whereas `printf()` writes data to the standard output device.

**Q. 22) Why should I prototype a function?**

**Answer:** A function prototype tells the compiler what kind of arguments a function is looking to receive and what kind of return value a function is going to give back. This approach helps the compiler ensure that calls to a function are made correctly and that no erroneous type conversions are taking place.

**Q. 23) How to write a program in c to print its own code?**

**Answer:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    FILE *fp;
    fp=fopen("prog.c","r");
    if(fp)
        while((ch=fgetc(fp))!=EOF)
            printf("%c",ch);

    else
        printf("
unable to open file");
    getch();
}
```

**Q. 24) What is the difference between `goto` and `longjmp()` and `setjmp()` ?**

**Answer:** A `goto` statement implements a local jump of program execution, and the `longjmp()` and `setjmp()` functions implement a nonlocal, or far, jump of program execution.

Generally, a jump in execution of any kind should be avoided because it is not considered good programming practice to use such statements as `goto` and `longjmp` in your program.

A `goto` statement simply bypasses code in your program and jumps to a predefined position. To use the `goto` statement, you give it a labeled position to jump to. This predefined position must be within the same function. You cannot implement `goto` between functions.

When your program calls `setjmp()`, the current state of your program is saved in a structure of type `jmp_buf`. Later, your program can call the `longjmp()` function to restore the program's state as it was when you called `setjmp()`. Unlike the `goto` statement, the `longjmp()` and `setjmp()` functions do not need to be implemented in the same function.

However, there is a major drawback to using these functions: your program, when restored to its previously saved state, will lose its references to any dynamically allocated memory between the `longjmp()` and the `setjmp()`. This means you will waste memory for every `malloc()` or `calloc()` you have implemented between your `longjmp()` and `setjmp()`, and your program will be horribly inefficient. It is highly recommended that you avoid using functions such as `longjmp()` and `setjmp()` because they, like the `goto` statement, are quite often an indication of poor programming practice.

#### Q. 25) Is using `exit()` the same as using `return`?

**Answer:** No. The `exit()` function is used to exit your program and return control to the operating system. The `return` statement is used to return from a function and return control to the calling function. If you issue a `return` from the `main()` function, you are essentially returning control to the calling function, which is the operating system. In this case, the `return` statement and `exit()` function are similar.

#### Q. 26) What are the advantages of the functions?

**Answer:** Debugging is easier. It is easier to understand the logic involved in the program. Testing is easier. Recursive call is possible. Irrelevant details in the user point of view are hidden in functions. Functions are helpful in generalizing the program.

#### Q. 27) How would you use the functions `sin()`, `pow()`, `sqrt()`?

**Answer:** Before using these functions `<math.h>` should be included in the program  
`sin(d)` will returns the sine of `d`, `pow(a,b)` will returns a the value `a` to the power `b` ( $a^b$ )  
eg: `pow(2, 3)` will returns 8; `sqrt(a)` returns the square root of `a` eg: `sqrt(4)` returns 2;

**Q. 28) How would you use the functions `memcpy()`, `memset()`, `memmove()` ?**

**Answer:** `memcpy()` function copies `n` bytes from the object pointed to by `s2` into the object pointed to by `s1`. If copying takes place between objects that overlap, the behavior is undefined.

`memmove()` function shall copy `n` bytes from the object pointed to by `s2` into the object pointed to by `s1`. Copying takes place as if the `n` bytes from the object pointed to by `s2` are first copied into a temporary array of `n` bytes that does not overlap the objects pointed to by `s1` and `s2`, and then the `n` bytes from the temporary array are copied into the object pointed to by `s1`.

`memset()` function copies `c` (converted to an unsigned char) into each of the first `n` bytes of the object pointed to by `s`. SYNTAX:

```
void *memcpy(void *s1, const void *s2, size_t n);
void *memmove(void *s1, const void *s2, size_t n);
void *memset(void *s, int c, size_t n);
```

**Q. 29) Is it possible to execute code even after the program exits the `main()` function?**

**Answer:** The standard C library provides a function named `atexit()` that can be used to perform cleanup operations when your program terminates.

You can set up a set of functions you want to perform automatically when your program exits by passing function pointers to the `atexit()` function.

**Q. 30) What are returned by `printf()`, `scanf()` functions, if they return anything means what are that?**

**Answer:** Return type of `printf()` and `scanf()` is integer. `scanf()` returns the no. of variables used and `printf()` returns the total no. of bytes.

**Q. 31) What is the purpose of `main()` function ?**

**Answer:** The function `main()` invokes other functions within it. It is the first function to be called when the program starts execution. It is the starting function. It returns an int value to the environment that called the program. Recursive call is allowed for `main()` also. It is a user-defined function.

Program execution ends when the closing brace of the function `main( )` is reached.  
It has two arguments

- 1) Argument count.
- 2) Argument vector (represents strings passed).

Any user-defined name can also be used as parameters for `main( )` instead of `argc` and `argv`.

**Q. 32) How to use the `cprintf`, `cscanf`, `sscanf`, `sprintf`, `vsscanf`, `vsprintf`, `vscanf` & `vprintf`?**

**Answer:**

`cprintf`: This function sends the formatted output to the text window on the screen.

`cscanf`: This function reads reads from the standard input device directly,avoiding buffering both by DOS and by the library.

`sprintf`: This function sends formatted output to the string.

`sscanf`: This function scans formatted text from the string and stores i on the variables pointed to by the arguments.

`vprintf`: This function sends formatted output to `stdin`,using an argument list.

`vscanf`: This function scans formatted text from `stdin` and stores it in the variables pointed to by the arguments.

`vsprintf`:This function sends formatted output to the string using argument list.

`vsscanf`: This function scans formatted text from the string and stores it in the variables.

**Q. 33) What is the code for `clrscr()` function?**

**Answer:** `clrscr()` is a function which creates a screen full of black dots which seems us to be clearing the screen or making a new screen.

**Q. 34) What do the functions `atoi()`, `itoa()` and `gctv()` do?**

**Answer:** `atoi()` This converts strings, like "23" or even "29dhjds" into integers (returning 23 and 29 respectively in this case).

`atoi` requires one `char *` argument and returns an `int` (not a float!).

If the string is empty, or first character isn't a number or a minus sign, then `atoi` returns 0.

If `atoi` encounters a non-number character, it returns the number formed up until that point.

### Q. 35) How do you use a pointer to a function?

**Answer:** The hardest part about using a pointer-to-function is declaring it. Consider an example. You want to create a pointer, `pf`, that points to the `strcmp()` function.

The `strcmp()` function is declared in this way:

```
int strcmp(const char *, const char * )
```

To set up `pf` to point to the `strcmp()` function, you want a declaration that looks just like the `strcmp()` function's declaration, but that has `*pf` rather than `strcmp`:

```
int (*pf)( const char *, const char * );
```

After you've gotten the declaration of `pf`, you can `#include <string.h>` and assign the address of `strcmp()` to `pf`: `pf = strcmp;`

### Q. 36) Write a program with out using `main()` function?

**Answer:**

```
#include<stdio.h>
#define decode(s,t,u,m,p,e,d) m##s##u##t
#define begin decode(a,n,i,m,a,t,e)
int begin()
{
printf(" hello ");
}
```

### Q. 37) Difference between Function to pointer and pointer to function

**Answer:** just go through the two examples written below, to clarify this doubt ex of pointer to function:

```
int (*function name)(Argument1,Argument2..)
```

The above declaration explains that its an pointer to a function whose return type is an integer.

ex of function to pointer :

```
int *function name(Argument1,Argument2..)
```

The above declaration explains that a function returns a pointer to an integer quantity.

**Q. 38) How do you determine whether to use a stream function or a low-level function?**

**Answer:** Stream functions such as `fread()` and `fwrite()` are buffered and are more efficient when reading and writing text or binary data to files. You generally gain better performance by using stream functions rather than their un-buffered low-level counterparts such as `read()` and `write()`.

In multi-user environments, however, when files are typically shared and portions of files are continuously being locked, read from, written to, and unlocked, the stream functions do not perform as well as the low-level functions. This is because it is hard to buffer a shared file whose contents are constantly changing. Generally, you should always use buffered stream functions when accessing non-shared files, and you should always use the low-level functions when accessing shared files.

**Q. 39) In C, `main()` is a function and where is defined `main()` in C. because every function has three parts.**

- 1) declaration
- 2) definition.
- 3) calling

**Answer:** Declaration is not needed if method is defined before calling. `main()` method is called by the OS when the program is run. So, it has only a definition..

**Q. 40) How do you use a pointer to a function ?**

**Answer:** The hardest part about using a pointer-to-function is declaring it. Consider an example. You want to create a pointer, `pf`, that points to the `strcmp()` function. The `strcmp()` function is declared in this way:

```
int strcmp(const char *, const char * )
```

To set up `pf` to point to the `strcmp()` function, you want a declaration that looks just like the `strcmp()` function's declaration, but that has `*pf` rather than `strcmp`:

```
int (*pf)( const char *, const char * );
```

After you've gotten the declaration of `pf`, you can `#include` and assign the address of `strcmp()` to `pf`: `pf = strcmp;`

**Q. 41) Is it better to use a macro or a function?**

**Answer:** The answer depends on the situation you are writing code for. Macros have the distinct advantage of being more efficient (and faster) than functions, because their corresponding code is inserted directly into your source code at the point where the macro is called.

There is no overhead involved in using a macro like there is in placing a call to a function. However, macros are generally small and cannot handle large, complex coding constructs. A function is more suited for this type of situation. Additionally, macros are expanded inline, which means that the code is replicated for each occurrence of a macro. Your code therefore could be somewhat larger when you use macros than if you were to use functions.

Thus, the choice between using a macro and using a function is one of deciding between the tradeoff of faster program speed versus smaller program size. Generally, you should use macros to replace small, repeatable code sections, and you should use functions for larger coding tasks that might require several lines of code.

**Q. 42) What is a static function?**

**Answer:** Static Function is a function which is only used in the file where it is declared and defined. Other files cannot use that particular function.

**Q. 43) What is a method?**

**Answer:** Method is a way of doing something, especially a systematic way; implies an orderly logical arrangement (usually in steps).

**Q. 44) When function say `abc()` calls another function say `xyz()`, what happens in stack?**

**Answer:** When some function `xyz()` calls function `abc()`. all the local variables, static links, dynamic links and function return value goes on the top of all elements of function `xyz()` in the stack. When `abc()` exit it's return value has been assigned to `xyz()`.

**Q. 45) What is the function of c?**

**Answer:** C is a good programming language . C functions are c language is used for providing software drivers for so many devices.

#### Q. 46) Advantages of a macro over a function ?

**Answer:** Macro gets to see the Compilation environment, so it can expand `__TIME__` `__FILE__` `#defines`. It is expanded by the preprocessor.

For example, you can't do this without macros.

```
#define PRINT(EXPR) printf( "#EXPR =%d", EXPR)
PRINT( 5+6*7 ) // expands into printf("5+6*7=%d", 5+6*7 );
```

You can define your mini language with macros:

```
#define strequal(A,B) (!strcmp(A,B))
```

Macros are a necessary evils of life. The purists don't like them, but without it no real work gets done.

#### Q. 47) What is a function and built-in function?

**Answer:**

A large program is subdivided into a number of smaller programs or subprograms. Each subprogram specifies one or more actions to be performed for a large program. such subprograms are functions.

The function supports only static and extern storage classes. By default, function assumes extern storage class .functions have global scope. Only register or auto storage class is allowed in the function parameters. Built-in functions that predefined and supplied along with the compiler are known as built-in functions. They are also known as library functions.

#### Q. 48) What is the output of

```
void main()
{
    int a = (1,2,3);
    printf("%d",a);
}
```

**with reason.**

**Answer:** The answer is 3.

The value which is last in the bracket is assigned to a(the variable).If it would have been any other value except 3,then it would have been assigned to 'a'.