GFacts

1. G-Fact 1

In C language, sizeof( ) is an operator. Though it looks like a function, it is an unary operator.

2. G-Fact 2

To know the IP address(es) of a URL/website, nslookup can be used at the shell/command prompt (cmd.exe). It works on both types of operating systems i.e. Linux/Windows. For example, to know the IP address of our website, type nslookup www.geeksforgeeks.org at the shell/command prompt.

3. G-Fact 3

In ISO C, you can define main either to take no arguments, or to take two arguments that represent the command line arguments to the program, like this:

```
int main (int argc, char *argv[])
```

Other platform-dependent formats are also allowed by the C and C++ standards; for example, Unix (though not POSIX.1) and Microsoft Visual C++ have a third argument giving the program's environment, otherwise accessible through getenv in stdlib.h:

4. G-Fact 4

In C, function parameters are always passed by value. Pass-by-reference is simulated in C by explicitly passing pointer values.

5. G-Fact 5

A large proportion of programming languages are bootstrapped, including BASIC, C, Pascal, Factor, Haskell, Modula-2, Oberon, OCaml, Common Lisp, Scheme, and more.

References:
http://en.wikipedia.org/wiki/Bootstrapping_%28compilers%29
http://www.oopweb.com/Compilers/Documents/Compilers/Volume/cha03s.htm

6. G-Fact 6

The C standard C99 allows inline functions and variable-length-arrays. So following functions are valid in C99 compliant compilers.

Example for inline functions

```
inline int max(int a, int b)
{
  if (a > b)
    return a;
  else
    return b;
}

a = max (x, y);
/*
  This is now equivalent to
  if (x > y)
    a = x;
  else
    a = y;
*/
```

Example for variable length arrays

```
float read_and_process(int n)
{
    float   vals[n];

    for (int i = 0; i < n; i++)
        vals[i] = read_val();
    return process(vals, n);
}
```

References:
http://en.wikipedia.org/wiki/C99

http://en.wikipedia.org/wiki/Variable-length_array
http://en.wikipedia.org/wiki/Inline_function

7. G-Fact 7

"Pointer *arithmetic* and array *indexing* [that] are equivalent in C, pointers and arrays are *different"* – Wayne Throop

References:
http://c-faq.com/aryptr/aryptrequiv.html

8. G-Fact 8

To uniquely construct a Binary Tree, Inorder together with either Postorder or Preorder must be given (See this for details). However, either Postorder or Preorder traversal is sufficient to uniquely construct a Binary Search Tree. To construct Binary Search tree, we can get Inorder traversal by sorting the given Preorder or Postorder traversal. So we have the required two traversals and can construct the Binary Search Tree.

9. G-Fact 9

The number of structurally different Binary Trees with n nodes is Catalan number Cn = (2n)!/(n+1)!*n!

References:
http://mathworld.wolfram.com/BinaryTree.html

10. G-Fact 10

Enumeration constants (`enum` values) are always of type `int` in C, whereas they are distinct types in C++ and may have size different from that of `int`.

Source:
http://en.wikipedia.org/wiki/Compatibility_of_C_and_C%2B%2B

11. G-Fact 11

Following relationship holds in any n-ary tree in which every node has either 0 or n children.

L = (n-1)*I + 1

Where L is the number of leaf nodes and I is the number of internal nodes.

Proof:
The tree is n-ary tree. Assume it has T total nodes, which is sum of internal nodes (I) and leaf nodes (L). A tree with T total nodes will have (T – 1) edges or branches.

In other words, since the tree is n-ary tree, each internal node will have n branches contributing total of n*I internal branches. Therefore we have the following relations from the above explanations,

n*I = T – 1
L + I = T

From the above two equations, it is easy to prove that L = (n – 1) * I + 1.

Thanks to venki for providing the proof.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

12. G-Fact 12

In C, struct keyword must be used for declaring structure variables, but it is optional in C++.

For example, following program gives error in C and works in C++.

```
struct node {
   int x;
   node *next; // Error in C, struct must be there. Works in C++
};

int main()
{
   node a;  // Error in C, struct must be there. Works in C++
}
```

And following program works in both C and C++.

```
struct node {
   int x;
   struct node *next;  // Works in both C and C++
};

int main()
{
   struct node a;  // Works in both C and C++
}
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

13. G-Fact 13

const Behaviour in C and C++

In C, the const qualified identifiers will have external linkage, where as in C++ it will have internal linkage. For example,

In C++, the following statement

```
float const interest_rate = 9.25;
```

is implicitly defined as

```
static float const interest_rate = 9.25;
```

i.e. the scope of *interest_rate* is limited to the block in which it is defined.

In C, the above statement will have external linkage when defined at file scope, i.e. it will be visible outside the current translation unit (source file).

The internal linkage of const qualified variables have some advantages in C++. We will cover them in next article.

Thanks to Venki for writing the above fact. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

14. G-Fact 14

In C, a structure cannot have static members, but in C++ a structure can have static members.

For example, following program causes compilation error in C, but works in C++.

```
#include<stdio.h>

struct test {
  static int i;  // Error in C, but works in C++.
};

int main()
{
  struct test t;
  getchar();
  return 0;
}
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

15. G-Fact 15

Atomic Operation

What is an atomic operation? An idea of atomic operation helps in understanding reentrancy, critical section, thread safety, synchronization primitives, etc… (we will have upcoming articles on each).

Atomicity, Atomic Operation:

In simple terms, atomicity is unbreakability, i.e. an uninterrupted operation. If two users issue a print command, each print should go in single attempt. If the printer driver is sending parts of data from two users, the printout will not be as expected. Hence, the printer driver must send the print command as unbreakable operation from one application at a time (in other words *synchronize* the access to printer).

Note that the data base terminology on atomicity would be different, yet the concept is same.

With an example we can understand the atomicity in programming well. Consider in a multi-threaded application, a function is incrementing a global/static variable,

count++; // count has permanent storage in RAM

The above statement can be decomposed into, atleast three operations.

1. Fetching *count* value
2. Incrementing *count* value
3. Storing the updated value

If a thread executing the function containing the above statement is fetching its value (say 2). It is possible that at this point of execution, the thread can be preempted and another thread may invoke the same function. Consequently, the value of *count* will be incremented to 3 by that thread. When the former thread is resumed, it still retains the previous value (2), instead of latest value (3), and ends up in writing back 3 again. Infact, the value of *count* should be 4 due to affect of both the threads.

Such kind of bugs are quite difficult to recreate and locate.

An example of atomic operation is instruction execution, usually an instruction feed to the execution unit can't be stopped in the middle. Yet, a statement in high level language results in multiple instructions. It is the root cause of non-atomic operations.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

16. G-Fact 16

Predict the output of following program.

```
#include <stdio.h>
int main()
{
   int x = 012;
   printf("%d",  x);
   getchar();
   return 0;
}
```

The program prints 10. **Putting a 0 before an integer constant makes it an octal number and putting 0x (or 0X) makes it a hexadecimal number.** It is easy to put a 0

by accident, or as a habit. The mistake is very common with beginners.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

17. G-Fact 17

Storage class of a variable determines whether the item has a global or local lifetime. In C, *typedef* is considered as a storage class like other storage classes (auto, register, static and extern), nevertheless the purpose of *typedef* is to assign alternative names to existing types.

For example, the following program compiles and runs fime

```
#include <stdio.h>
int main()
{
  typedef int points;
  points x = 5;
  printf("%d ", x);
  return 0;
}
```

Output:

```
5
```

But the following program fails with compiler error.

```
#include <stdio.h>
int main()
{
  typedef static int points;
  points x;
  return 0;
}
```

Output:

```
Compiler Error: multiple storage classes in declaration specifiers
```

See this quiz for practice on storage class and type specifiers. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

18. G-Fact 18

Finding nth Fibonacci Number using Golden Ratio:
We have discussed different methods to find nth Fibonacci Number.

Following is another mathematically correct way to find the same.

nth Fibonacci Number Fn = $\left\lfloor \frac{\varphi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor, \ n \geq 0.$

Here φ is golden ratio with value as $(\sqrt{5}+1)/2$.

The above formula seems to be good for finding nth Fibonacci Number in O(Logn) time as integer power of a number can be calculated in O(Logn) time. But this solution doesn't work practically because φ is stored as a floating point number and when we calculate powers of φ, important bits may be lost in the process and we may get incorrect answer.

References:
https://www.youtube.com/watch?v=-EQTVuAhSFY
http://en.wikipedia.org/wiki/Fibonacci_number

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.