C++ Test II

Author: Adrian Michałek
Contact: devmichalek@gmail.com

Difficulty: All levels

Task 0. ***(1P)***

This is an example task. The further tasks will look alike. As you can see below is a part of code to examine. Each task contains description and questions to answer. Answers to these questions are always one page further so there is no reason to scroll down and search for them. You can write your answers on the paper and quickly check if you are right by checking the next page. Each task contains available number of points that you can get. For instance if you correctly answer the question for this task you get ***(1P) – 1 point*****.** There is no reason to test your knowledge if you feel tired with the test. Feel free to finish this test at any moment and check how many points you've got. Let's start with the first task.

The following code was compiled and executed.

```
#include <stdio.h>

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

What was printed to the standard output? ***(1P)***

Correct answer is:

```
Hello world!
```

(1P)

Task 1. ***(1P)***

```
#include <stdio.h>

int main()
{
	double data[7][4];

	return 0;
}
```

Write a variable declaration. Variable should be a pointer that points to `double` array 7x4. ***(1P)***

Correct answer is:

```
#include <stdio.h>

int main()
{
	double data[7][4];
	double (*pointer)[7][4] = &data;
	return 0;
}
```

(1P)

Task 2. ***(1P)***

Look at the implemention of the singleton:

```
class Singleton
{
public:
    static Singleton& getInstance()
    {
        static Singleton instance;
        return instance;
    }

private:
    Singleton() {}
};

int main()
{
    return 0;
}
```

What is missing in the above `class`? Finish the `Singleton class`. ***(1P)***

Perfect singleton should be created only once. Singleton should disallow to copy itself:

```
class Singleton
{
public:
    static Singleton& getInstance()
    {
        static Singleton instance;
        return instance;
    }

private:
    Singleton() {}

public:
    Singleton(Singleton const&) = delete;
    void operator=(Singleton const&) = delete;
};

int main()
{
    return 0;
}
```

(1P)

Task 3. ***(2P)***

The following code was compiled and executed:

```c
#include <stdio.h>
#include <stdarg.h>

void foo(int n, ...)
{
    va_list vl;
    va_start(vl, n);
    char c = va_arg(vl, char);
    float f = va_arg(vl, float);
    printf("char %c, float %f\n", c, f);
    va_end(vl);
}

int main()
{
    foo(2, 'x', 12.345f);
    return 0;
}
```

1. Variadic function has been called. What was printed to the standard output and why? ***(1P)***

2. Correct function `foo` so that it correctly prints passed variables. ***(1P)***

1. The following string was redirected to the console:

char x, float -36893488147419103232.000000

Parameters of functions that correspond to … are promoted before they are passed to variadic function. `char` and `short` are promoted to `int`, `float` is promoted to `double` etc.

The reason for this is that early versions of C did not have function prototypes; parameter types were declared at the function site but were not known at the call site. But different types are represented differently, and the representation of the passed argument must match the called function's expectation. So that char and short values could be passed to functions with int parameters, or float values could be passed to functions with double parameters, the compiler "promoted" the smaller types to be of the larger type. This behavior is still seen when the type of the parameter is not known at the call site.

(1P)

2. Solution is to correct types passed to `va_arg` macro:

```c
#include <stdio.h>
#include <stdarg.h>

void foo(int n, ...)
{
    va_list vl;
    va_start(vl, n);
    char c = va_arg(vl, int);
    float f = va_arg(vl, double);
    printf("char %c, float %f\n", c, f);
    va_end(vl);
}

int main()
{
    foo(2, 'x', 12.345f);
    return 0;
}
```

The output for the above code is now:

char x, float 12.345000

(1P)

Task 4. ***(1P)***

The following code won’t compile:

```
#include <iostream>

class Simple
{
public:
    template<const char* message>
    void calculate()
    {
        std::cout << message << std::endl;
    }
};

int main()
{
    Simple simple;
    simple.calculate<"Message">();
    return 0;
}
```

Compiler gives the following error „a template argument may not reference a non-external entity”.

Describe the problem with this template function. ***(1P)***

Function `calculate` would not be a useful utility. Since `const char*` message is is not of the allowed form of a template argument, it currently does not work.

Let's assume that kind of template arguments work. Because they are not required to have the same address for the same value used, you will get different instantiations even though you have the same string literal value in your code.

(1P)

Task 5. ***(1P)***

The following code was compiled and executed:

```
#include <iostream>

bool Test()
{
    std::cout << "Test" << std::endl;
    return false;
}

int main()
{
    bool result = Test;
    std::cout << result << std::endl;
    return 0;
}
```

What was printed to the console and why? ***(1P)***

The following string was printed to the console:

```
1
```

Explanation:

You can regard `Test` as the address of the function. Crucially, it will have a non-zero numerical value, and a statement consisting of a single variable is grammatically correct.

(1P)

Task 6. ***(1P)***

The following code was compiled:

```
#include <iostream>

void callme(int* a, int* b)
{
        static int x = 0;
        ++x;
        a = &x;
        static int y = *b;
        ++y;
        b = &y;
}

int main()
{
        int i = 5;
        int* a = nullptr;
        int* b = &i;
        callme(a, b);
        return 0;
}
```

After the program was executed what values were stored in pointer a and b after function call `callme`? ***(1P)***

a is equal to nullptr
b points to the address of i

(1P)

Task 7. ***(1P)***

The following code was compiled and executed:

```cpp
#include <iostream>

int main()
{
	char a = 30, b = 40, c = 10;
	char d = (a * b) / c;
	char e = 255, f = 5;
	char g = e + f;
	std::cout << "d = " << static_cast<int>(d) << std::endl;
	std::cout << "g = " << static_cast<int>(g) << std::endl;
	return 0;
}
```

What was printed to the standard output? ***(1P)***

Correct answer is:

```
120
4
```

Explanation:

At first look, the expression (a*b)/c seems to cause arithmetic overflow because signed characters can have values only from -128 to 127 (in most of the C compilers), and the value of subexpression '(a*b)' is 1200 which is greater than 128. But integer promotion happens here in arithmetic done on char types and we get the appropriate result without any overflow.

Some data types like `char`, `short int` take less number of bytes than `int`, these data types are automatically promoted to int or `unsigned int` when an operation is performed on them. This is called **integer promotion**. For example no arithmetic calculation happens on smaller types like `char`, `short` and enum. They are first converted to `int` or `unsigned int`, and then arithmetic is done on them. If an `int` can represent all values of the original type, the value is converted to an `int`. Otherwise, it is converted to an `unsigned int`.

(1P)

Task 8. ***(1P)***

Explain what does the `extern "C"` mean. ***(1P)*** `extern "C"` makes a function-name in C++ have 'C' linkage (compiler does not mangle the name) so that client C code can link to (i.e use) your function using a 'C' compatible header file that contains just the declaration of your function. Your function definition is contained in a binary format (that was compiled by your C++ compiler) that the client 'C' linker will then link to using the 'C' name.

Since C++ has overloading of function names and C does not, the C++ compiler cannot just use the function name as a unique id to link to, so it mangles the name by adding information about the arguments. A C compiler does not need to mangle the name since you can not overload function names in C. When you state that a function has extern "C" linkage in C++, the C++ compiler does not add argument/parameter type information to the name used for linkage.

(1P)

Task 9. ***(1P)***

Write a function declaration.

Function takes no arguments and returns a pointer to the char array with the length of 5. ***(1P)***

Correct answer is:

```
char(*(f)(void))[5];
```

(1P)

Task 10. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 11. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 12. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 13. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 14. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 15. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 16. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

```
Hello world!
```

(1P)

Task 17. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 18. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

```
Hello world!
```

(1P)

Task 19. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

```
Hello world!
```

(1P)

Task 20. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 21. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 22. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 23. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 24. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 25. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 26. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 27. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 28. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 29. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

```
Hello world!
```

(1P)

Task 30. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 31. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 32. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 33. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 34. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 35. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 36. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 37. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 38. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 39. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 40. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 41. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 42. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 43. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 44. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 45. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

```
Hello world!
```

(1P)

Task 46. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 47. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 48. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 49. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Task 50. ***(1P)***

Description:

Question? ***(1P)***

Correct answer is:

Hello world!

(1P)

Thank you for trying your best with my test!
If you found any problems, flaws or if you want to give a comment contact me:
devmichalek@gmail.com