Ответы на вопросы экзамена по курсу «Языки программирования» 10.01.2018

В ответах курсивом выделены необязательные пояснения, которые можно опустить (особенно на экзамене)

Вариант 2

Задача 2-1

Замените знаки вопроса (`?????`) в приведенном ниже фрагменте программе на языке C++ так, чтобы в стандартный вывод выдавалась сумма эдементов массива `a`. Никакой код куда-либо еще добавлять нельзя.

```
int x = 0; int a[] = { 1,3,5,7,9 }; size_t n = sizeof(a)/sizeof a[0];
std::for_each(a, a + n, ?????);
std::cout << x << std::endl;
```

Ответ

Нужна лямбда-функция **`[&x](int y) { x += y; }`**
Не забывать про замыкание по ссылке (x можно опустить, но ссылка обязана остаться)

Задача 2-2

Пусть s1 и s2 описаны как переменные строкового типа (string), им присвоены непустые (длина >= 1) строки.. Можно ли утверждать, что длина конкатенации строк (s = s1 + s2) будет равна сумме длин s1 и s2 для любого языка программирования? Варианты языков: C++ (s.size()), Java (s.length), Go (len(s)), Swift (s.count), C# (s.Length), Python (len(s)). В скобках для каждого языка указаны функции или свойства, возвращающие длину строки.

Ответ

См. ответ на задачу 1-2 из первого варианта. Тут важно понимать, что в любом языке операция конкатенации просто дописывает символы второго операнда в конец первого. Если у нас сливаются на границе операндов композитные символы, то единственный язык, который учитывает эту специфику, - Swift. В остальных четко соблюдается правило: длина (s1+s2) = длина (s1) + длина (s2). Нормализации текста не происходит. Пример на Go:

```
package main


import (
        "fmt"
)


func main() {
        s := "cafe"
```

```
        s += "\u0301" // adding acute character after e
        fmt.Print(len(s))
        fmt.Println("->" + s)
        s = "caf"
        s += "\u00e9" // adding composite é
        fmt.Print(len(s))
        fmt.Println("->" + s)
}
```

Внешне напечатано одно и то же, но с разной длиной:

6->café
5->café

А вот аналогичная Swift-программа дала бы один и тот же результат:
4->café
4->café

Задача 2-3

Объясните, допустимо ли присваивание **`x = y`** в следующем фрагменте программы на C#. Если допустимо, то какие проблемы с **`x`** и **`y`** могут возникнуть в дальнейшем при выполнении программы?

```
class Foo { public void foo () {} }
class Bar: Foo { public void bar () {} }
object [] x; Bar [] y = new Bar[10];
x = y;
```

Ответ

Язык C# допускает такое присваивание. То, что после него нельзя вызывать **`x.foo()`** - это не проблема, а свойство любого ОО-языка. Проблема в том, что **`x`** нельзя модифицировать так, как это следовало бы из его объявления. Ведь если забыть об этом присваивании, то можно в **`x`** положить **<u>любой</u>** объект. Это ведь массив из просто объектов. Но тут надо вспомнить об объектно-референциальной модели языка C#. После присваивания **`x`** ссылается на объект-массив, который является ОДНОРОДНЫМ и состоит только из объектов типа Bar или производных от него, поэтому попытка положить в него что-то другое (**`x[0] = new Object()`** или **`x[0] = new Foo()`**) приводит к «аварии» - возбуждению исключения **`System.ArrayTypeMismatchException`**.
Заметим, что для динамических массивов `List<T>` принята логичная точка зрения, что списки из разных (неважно, ковариантных или контравариантных) типов всегда инвариантны.
Замечание 2: задача полностью аналогична задаче из первого варианта, поскольку языки ведут себя в данном случае одинаково

Задача 2-4

Написать на языке C++ шаблонную функцию **`MakeWord`** с переменным числом параметров (как своих, так и шаблонных), которая возвращает целое число, являющееся побитовой

суммой (операция суммирования — побитовое «или» |) своих параметров. Аргументы шаблона должны быть типами.
Например `std::cout << MakeWord(1,2);` должно выдать 3, а
`std::cout << MakeWord(2,4,9);` должно выдать 15.

Ответ

Тут **ОБЯЗАТЕЛЕН** вариадический шаблон.

C++17:

```
template <typename T, typename ... Bits> int MakeWord(T&& bit, Bits&& ...bits)
{
    return (bit | ... | bits); // скобки обязательны!!!
}
```

Задача 2-5

Являются ли реализации обобщенных классов на языке C# более эффективными, чем их необобщенные аналоги. Например, будет ли реализация обобщенного класса Stack<int> более эффективной, чем реализация класса Stack из Object?

Ответ

Да, будет. В частности, для стека это происходит за счет отсутствия неявных операций упаковки в объект типов значений. В конечном счете, компилятор генерирует отдельный код для конкретизации типом значений, а также для конкретизации ссылочными типами. Таким образом, если в программе есть конкретизации `Stack <double>`, `Stack<int>`, `Stack<String>`, `Stack<char[]>`, то будет сгенерировано три набора функций членов: по одному набору для каждого типа значений (`double`, `int`) и один набор для ссылочных типов (`String`, `char[]`). Отметим, что возможность генерировать один и тот же код для типов-ссылок связана с тем, что с точки зрения стека типы отличаются только размером. Если размер типов одинаков, то код функций, копирующих, значения этого типа, одинаков. Каждый набор сгенерированных методов не использует функции упаковки/распаковки и проверяемые преобразования типов (правильность соответствия типов уже проверена компилятором). Это дает прирост эффективности по сравнению с универсальным типом.
Правда, код несколько разрастается, но это несущественно, поскольку экономия на коде упаковки и распаковки позволяет нивелировать это разрастание.

Задача 2-6

Напишите какой-нибудь пример сопрограммы на языке Python. Чем сопрограммы языка Python отличаются от go-программ языка Go?

Ответ

```
def corr():
    i = (yield)
    print(i)
    yield i + 1
```

```
c = corr()


c.send(None)
print(c.send(1))
```

Сопрограммы Python (**не путать с сопрограммами Кнута!!!**) не просто генераторы, а генераторы, которые ПОЛУЧАЮТ информацию от вызывающей программы с помощью yield-выражения (в сопрограмме) и метода send (в вызывающей программе):
x = (yield)
Не путайте его с yield-оператором из примера, который ОТДАЕТ информацию (вместе с управлением) в вызываемую программу. Сопрограммы Python однопоточны (*желающие могут почитать про Global Interpreter Lock*). Есть и асинхронные расширения современного Питона, но мы их на лекциях не рассматривали.
Замечание: удивительно много народа вместо сопрограмм писало пример генератора.

Отличия от го-программ: го-программы выполняются асинхронно, хотя могут и использовать один и тот же поток. Главное отличие состоит в том, что го-программы не возвращают никакой информации в программу после завершения и не передают управление в вызывающую программу. Вместо этого го-программы передают информацию и синхронизируются через каналы.

Задача 2-7

Дайте определение словаря (отображения, таблицы). В каких языках из списка : Python, C++, C# - словари входят в базис языка, а в каких — входят в стандартную библиотеку? Для каждого языка дайте пример (инициализации и обращения к элементу) словаря. Почему в JavaScript и PHP такого понятия нет ни в базисе, ни в стандартную библиотеке?

Ответ

Таблица (словарь) — это контейнер, который хранит пары «ключ-значение» и позволяет эффективно отыскивать значение по ключу (логарифмическое время или быстрее для хэшей) и достаточно эффективно вставлять и удалять ключи.
В Python, как и в других динамических языках словарь входит в базис языка, а в C# и C++ - в стандартную библиотеку.
Python:

```
dict = {"key1" : "value1", "key2" : "value2"}
print(dict["key1"])  # value1
```

C#:

```
            IDictionary<int, string> dict = new Dictionary<int, string>();
            dict.Add(1, "value1");
            dict.Add(2, "value2");
            string v = dict[1]; // value1
```

C++:

```
       std::map<int, std::string> map;
       map[1] = "value1";
       map[2] = "value2";
       std::string v = map[1]; // value1
```

В JavaScript нет нужды в таком контейнере, поскольку любой объект сам является таблицей:
var obj = {}
obj["prop"] = value // то же самое, что obj.prop = value

А в PHP любой массив является таблицей.

Задача 2-8

В приведенной ниже программе на языке Go есть ошибка, которая проявляется во время выполнения. В чем она заключается, и как ее исправить (указание — надо закомментировать одну строку)?
Что будет выдано после исправления?

```
package main;import "fmt"

func b(c chan int, quit chan int) {
      go func() {
            for i := 10; i >= 0; i-- {
                  c <- i
            }
            quit <- 0
      }()
}
func main() {
      quit := make(chan int)
      c := make(chan int)
      b(c, quit)
      for i := 10; i >= 0; i-- {
        fmt.Print(<-c);fmt.Print(" ");
      }
      fmt.Print(<-quit)
      select {
      case c <- 1: // do nothing
      case <-quit:
            return
      }
}
```

Ответ

Ошибка состоит в том, образуется «тупик». Анонимная го-программа из функции `b` пишет в канал `c` 10 чисел, далее пишет 0 в канал `quit`. Главная программа после запуска `b` читает из `c` 10 чисел и их печатает:

10 9 8 7 6 5 4 3 2 1 0
Далее `fmt.Print(<-quit)` печатает 0, считав его из `quit`. Теперь все каналы — пусты, и никто в них ничего не пытается записать. Поэтому главная программа висит на ожидании чтения. Типичный тупик. Надо убрать (закомментировать) `fmt.Print(<-quit)` . После этого программа завершится.

Замечание. Можно скопировать текст программы и запустить ее на сайте play.golang.org. Там можно проверить другие варианты ответов (и убедиться в их неправильности).