1. Дополнительные возможности

1.1. Парадигмы программирования

- Структурное (процедурное) программирование

Программа = алгоритм + данные

- Объектно-ориентированное программирование

Программа = совокупность объектов, взаимодействующих друг с другом

1.2. Дополнительные средства C++

- Конструкции языка
- Пространства имен
- Перегрузка функций
- Шаблоны функций
- Обработка исключений
- Организация ввода/вывода

1.3. Конструкции языка

1. Определение констант

```
const  тип  имя = значение;
const int SZ = 80;
```

2. Цикл for

```
for (тип  имя = врж1;  врж2;  врж3)
    тело цикла
for(int i = 0; i < SZ; ++i)
    s += a[i];
```

1.3. Конструкции языка
(продолжение)

3. Определения в условиях

   if(*тип имя* = *врж*)

   . . .

4. Последовательность предложений языка

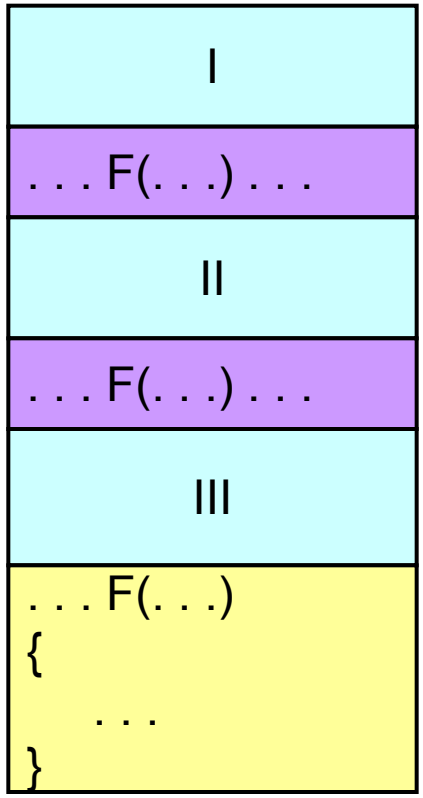
5. Встроенные функции

   inline *заголовок функции*

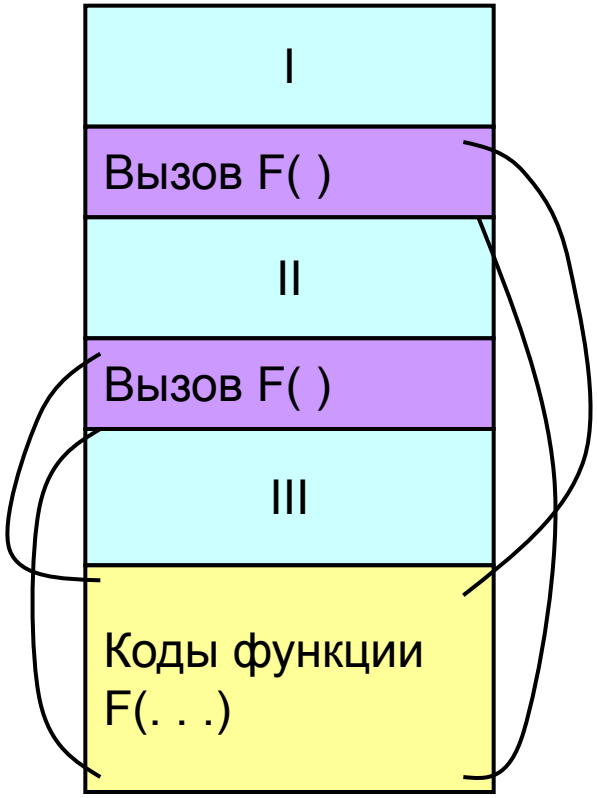
   тело функции

1.3. Конструкции языка
(продолжение)

Исходный текст



Обычная функция



Встроенная функция

1.3. Конструкции языка (продолжение)

5. Тип данных «ссылка»

тип & *имя*;

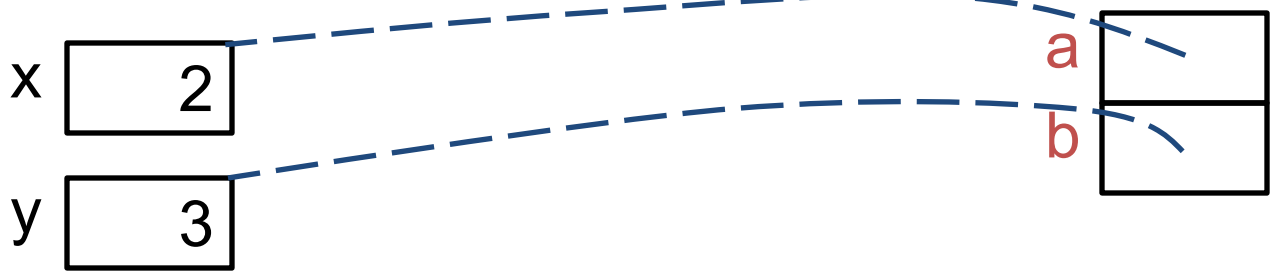
(a)
```
void swap(int *a, int *b)
{
    int t = *a; *a = *b; *b = t;
}
```

(b)
```
void swap(int &a, int &b)
{
    int t = a; a = b; b = t;
}
```

1.3. Конструкции языка
(продолжение)

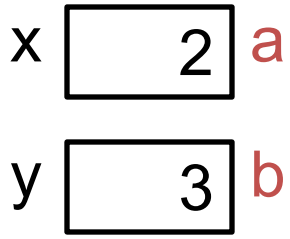
```
int x = 2, y = 3;
swap(&x, &y);
```

(a) `void swap(int *a, int *b);`




```
swap(x, y);
```

(b) `void swap(int &a, int &b);`



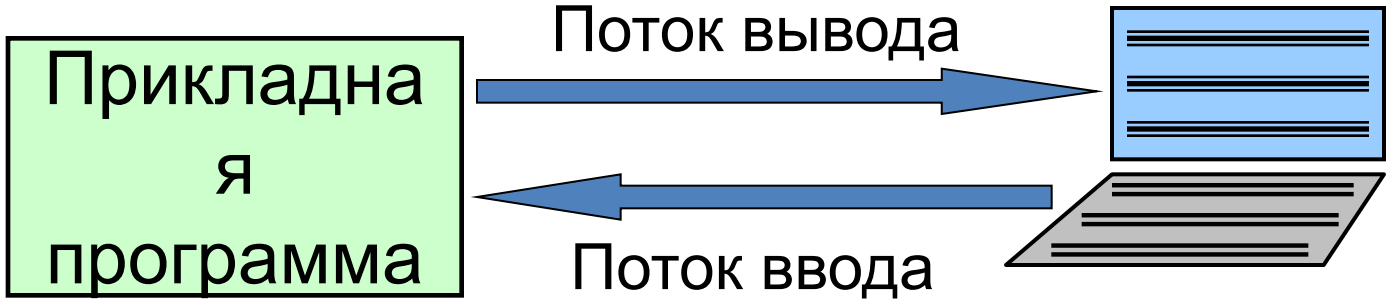
1.4. Пространства имен

- Пространство имен – **`namespace`**
- Библиотека C++ – в **`std`**

1.5. Организация ввода/вывода

Потоковый ввод/вывод

- Буферизация
- Форматирование
- Преобразование



1.5. Организация ввода/вывода (продолжение)

#include <iostream>

Ввод данных:

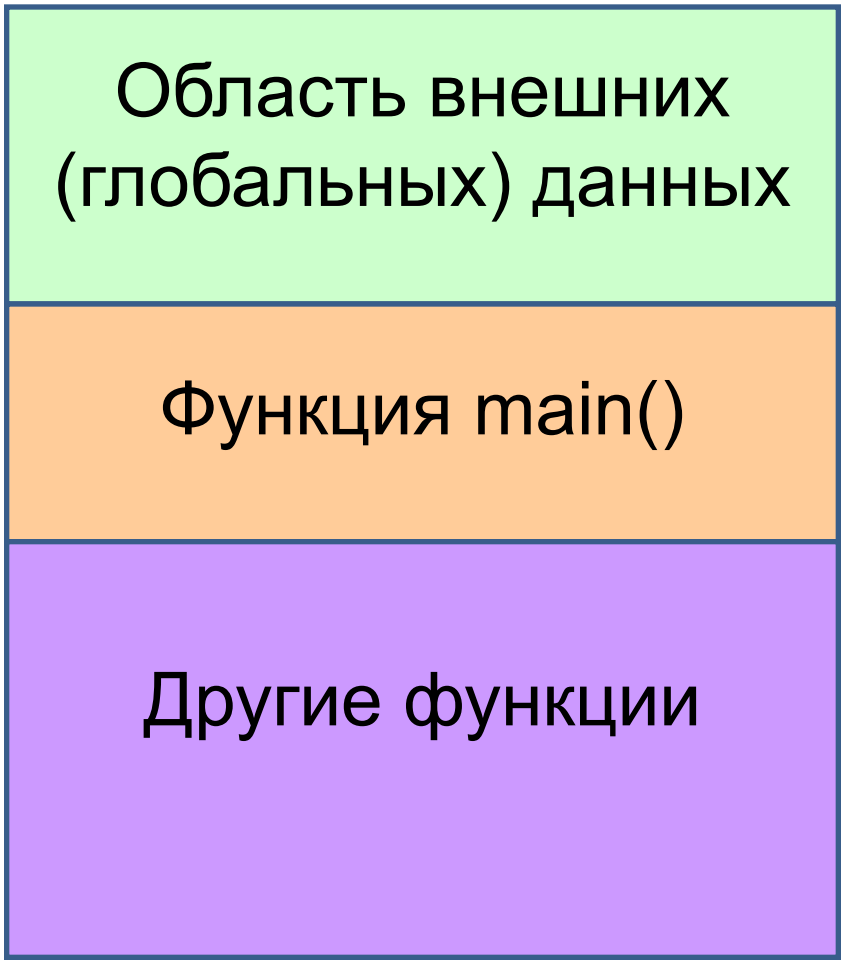
тип var1; . . .

std::cin >> var1 >> . . .;

Вывод данных:

std::cout << *врж1* << . . . << std::endl;

1.6. Структура программы на С

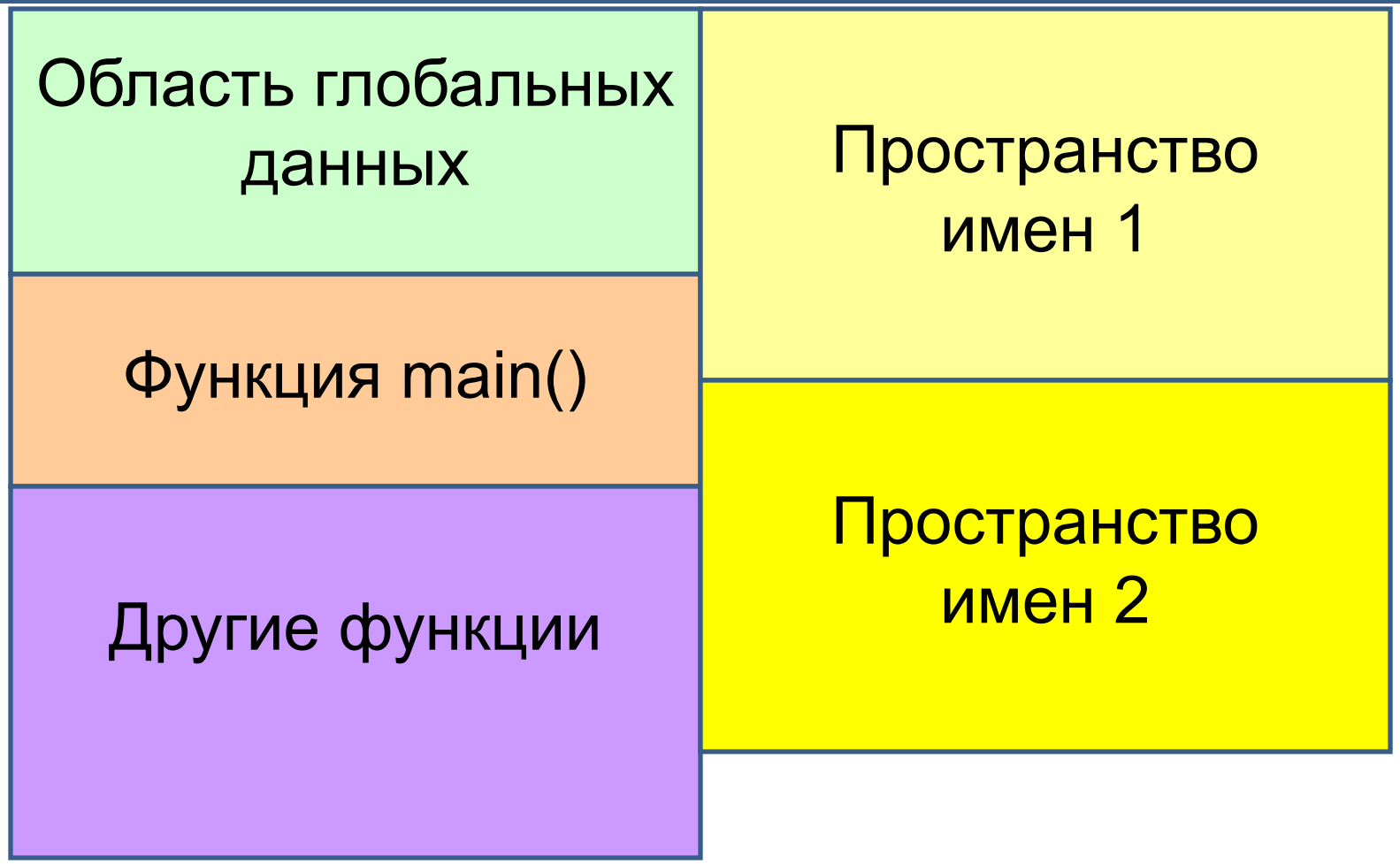


1.6. Структура программы на С
(продолжение)

```
// файл 1
int n;
static int p;
int main()
{
  n = 5;
}
void f1()
{
  int n = 15;
  ++n;
  p = 12;
}
```

```
// файл 2
extern int n;
extern int p;   → ???
void f2()
{
  n = 28;
}
void f3()
{
   p = 23;   → ошибка!
   static int n = 3;
   ++n;
}
```

1.7. Структура программы на C++



1.8. Структура пространства имен

1.9. Определение пространства имен

namespace *имя* **{**

определения

}

Пример:

```
namespace MyNameSpace {
   int i, k;
   void f(int j) { std::cout << (i + j); }
}
```

1.10. Доступ к объектам

- Внутри пространства имен – непосредственно
- Вне пространства имен – с помощью оператора разрешения видимости **::**

имя_пространства **::** *имя_члена*

Пример:

```
MyNameSpace::i = 10;
MyNameSpace::f(12);
```

1.10. Доступ к объектам
(продолжение)

- Использование псевдонимов

```
namespace MNS = MyNameSpace;
  MNS::i = 10;
  MNS::f(12);
```

1.11. Разделение пространства имен

```
namespace NS {
  int i;
  . . .
}
. . .
namespace NS {
  int j;
  . . .
}
```

1.12. Пример: стек символов

```
// файл stack.h : интерфейс стека
namespace Stack {
    void push(char );
    char pop();
}
```

1.12. Пример: стек символов
(продолжение)

```
// файл stack.cpp : реализация стека
#include “stack.h”
 namespace Stack {
      const int SZ = 200;
      char st [SZ];
      int top = 0;
      void push(char c) { … }
}
char Stack::pop() { … }
```

1.12. Пример: стек символов
(продолжение)

```
// файл user.cpp: использование стека
 #include “stack.h”
void f()
{
      Stack::push(‘c’);
      if(Stack::pop() != ‘c’)
          std::cout << “error” << std::endl;
}
```

1.13. Инструкция using

- **using namespace** *имя_пространства*;

  using namespace std;

- **using** *имя_пространства*::*имя_члена*;

  using std::cin;

1.14. Пример

1.15. Результат выполнения

3. In NS1::f: q = 19210;
2. in NS1: q = 123454321;
1. ::q = 555

1.16. Неименованное пространство

```
namespace {
  . . .
}
```

Создает идентификаторы, уникальные внутри данного файла. Вне данного файла идентификаторы недоступны

(Эквивалентно классу памяти **static** в C)

1.17. Перегрузка функций

Прототипы функций:

тип_результата имя (*список_параметров*);

Примеры:

```
void f(int);
void f(float);
void f(char [], int);
void f(const char *);
```

1.18. Использование функции

Вызов функции:

. . . *имя* (*список_аргументов*) . . .

```
f(12);
f("Hello");
```

1.19. Алгоритм выбора функции

1. Точное соответствие вызова функции прототипу
2. Преобразование аргументов – расширение (**`char`** к **`int`**, **`int`** к **`long`**, **`float`** к **`double`**)
3. Стандартное преобразование типа (**`int`** к **`float`** , **`long`** к **`int`** и др.)
4. Преобразования, требующие временных переменных (параметр – ссылка)
5. Преобразования, определенные пользователем

1.20. Примеры

Прототипы функций

```
void f(char);
void f(double);

void f(double, int);

void f(const int &, const
   char *);
```

Вызовы функций (на этапе компиляции)

```
int a; float c;
f('a'); // прав.1
f(c); // прав.2
f(12); // прав.3
f(c, 8); // прав.2
f(c, c); // прав.3
f(a, "*"); // прав.1
f(12, "*"); // прав.4
```

???

1.21. Параметры по умолчанию

Прототип функции:

```
char *fc(char, int = 1); // !!!
```

Эквивалентно

```
char *fc(char);
char *fc(char, int);
```

1.21. Параметры по умолчанию (продолжение)

Определение функции:

```
char *fc(char a, int b) // !!!
{
  . . .
}
```

Вызов функции … fc('*') …

эквивалентен … fc('*', **1**) …

1.22. Шаблоны функций

```
void swap(int &a, int &b)
{
  int t = a; a = b; b = t;
}
void swap(double &a, double &b)
{
  double t = a; a = b; b = t;
}
```

1.22. Шаблоны функций
(продолжение)

```
template <class MyType>
void swap(MyType &a, MyType &b)
{
  MyType t = a; a = b; b = t;
}
```

1.22. Шаблоны функций
(продолжение)

Вызов функции:

int a, b;

double c, d;

swap(a, b); □ генерируется код функции1

swap(c, d); □ генерируется код функции2

swap(a, b); □ используется код функции1

1.23. Обработка исключений

- Анализ и идентификация ошибки

```
int f(int &res )
{
   int errcode;
   . . .
   if (условие)
      диагностирование ошибки: errcode = 1
    . . .
     res = … ;
   return errcode;
}
```

1.23. Обработка исключений
(продолжение)

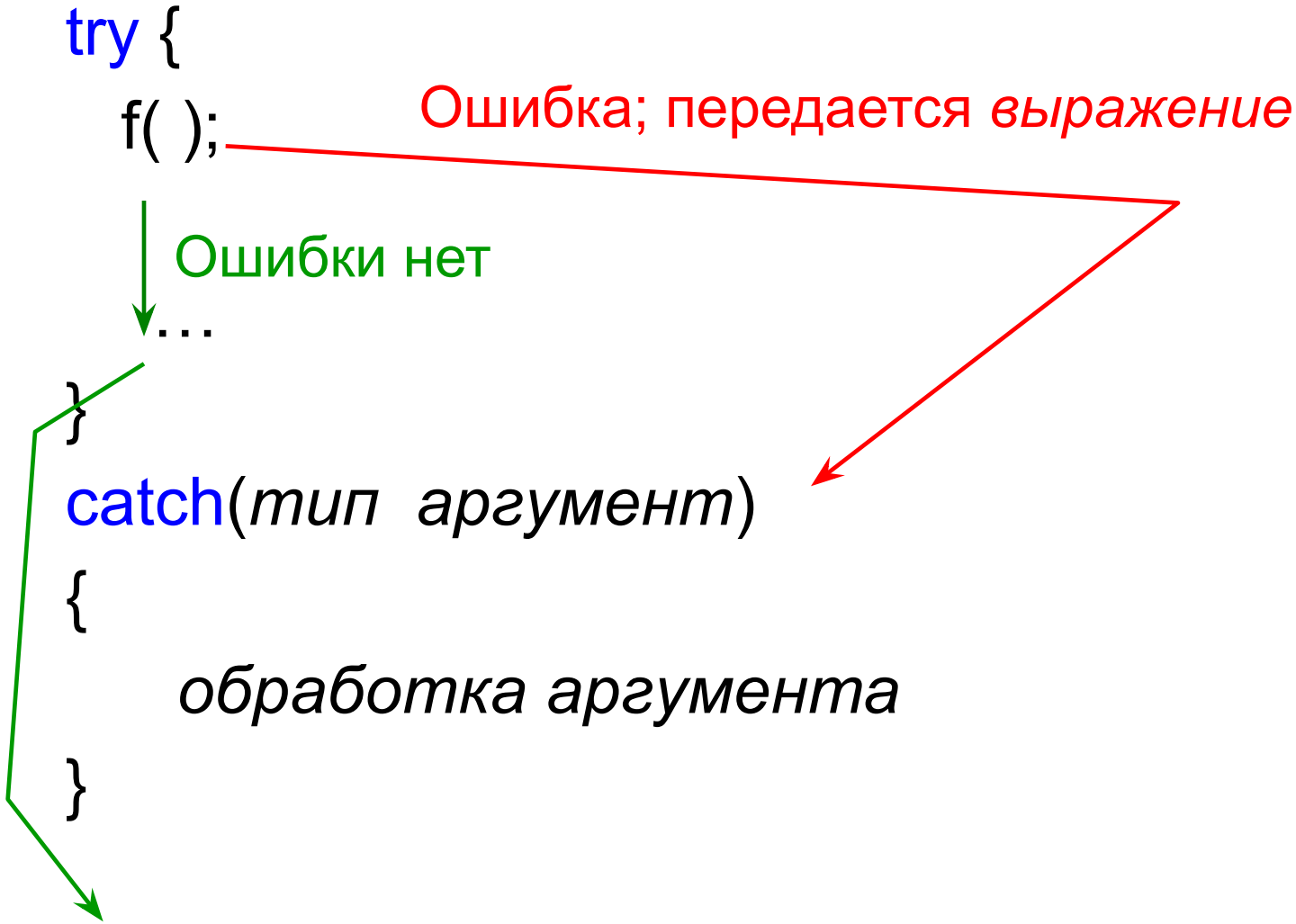
- Обработка ошибки

```
int rc, res;
. . .
rc = f(res);
if( rc == 1)
      обработка ошибки 1
   . . .
```

1.24. Диагностирование ошибки

```
int f( )
{
   int res;
   . . .
   if (условие)
      throw выражение;
    . . .
   return res;
}
```

1.25. Обработка ошибки



1.26. Пример

1.27. Результаты тестирования

string: "asdfvdewghu"; index = 8

g

string: "dfgrt"; index = 8

incorrect index