Введение в современные компьютерные технологии

Практикум 3

Дрюк Андрей

Немнюгин Сергей Андреевич

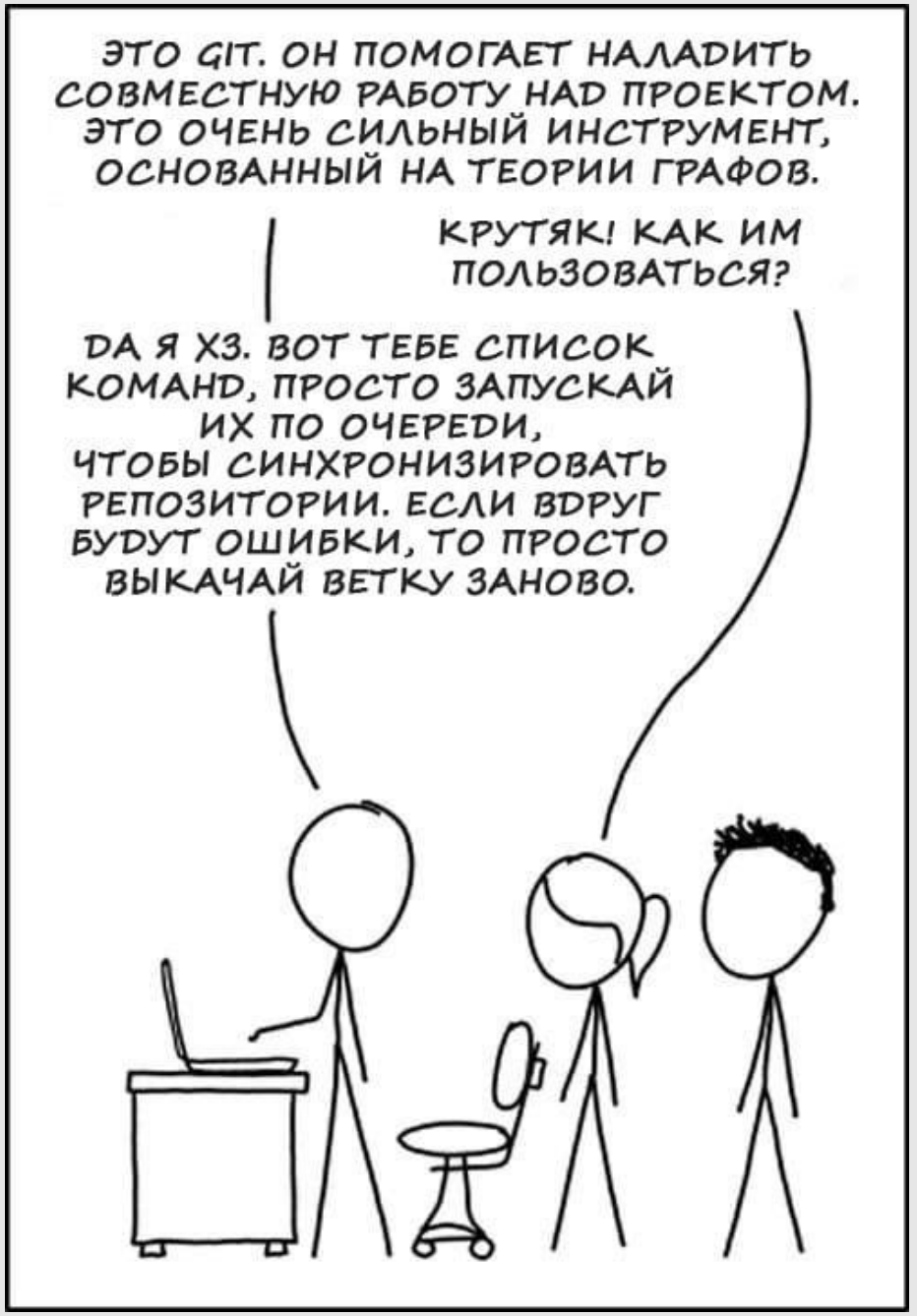
Санкт-Петербургский государственный университет

2022

Git

Git - программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение…

https://github.com/AndreiDriuk/Intro_in_mod_prog.git



Цикл for(.. ; .. ; ..){}

for(объявление переменных; условие; изменение счетчика){
    //тело цикла;
}

```
for(int i = 0; i < 10; ++i)
{
    cout<<i<<endl;
}
```

1. Если после for() или if() нет фигурных скобок, их действие распространяется только на одну строка после !!!!

while(){}

```
while (Условие) {
   Тело цикла;
}
```

```cpp
#include <iostream>

using namespace std;

int main() {
int i = 0; // инициализируем счетчик цикла.
int sum = 0; // инициализируем счетчик суммы.
while (i < 1000) {
   i++;
   sum += i; //sum = sum+i;
}
cout << "Сумма чисел от 1 до 1000 = " << sum << endl;
return 0;
}
```

do{...}while()

Do{…}
while (expression)

Выражение expression в операторе do-while вычисляется после выполнения тела цикла. Поэтому тело цикла всегда выполняется по крайней мере один раз!!!

Задание по while

1. **Создадим новый проект IntroInWhile**
2. **Написать приложение, которое будет запрашивать ввести пользователя число cin>>x**
3. **Программа выполняется, пока пользователь не в введет число больше 1000**

Управляющие операторы:

1. `continue`-прекратить итерацию [наиболее вложенного] цикла и начать следующую.
2. `break`-прекратить итерацию и завершить выполнение цикла.

```
int main()
{
    for (int i = 0; i < 10; ++i) {
        if (i % 2 == 0) {
            cout << i << endl;
        }
        else {
            cout << "odd" << endl;
            continue;
            cout << "You will not see that" << endl;
        }
    }

}
```

```
0
odd
2
odd
4
odd
6
odd
8
odd
```

Управляющие операторы, задание:

вывести на экран только простые числа в диапазоне от 1 до 100;

1. **Создадим переменную bool isPrime вне цикла, которая будет показывать, что число простое(true/1) или нет (false/0). Присвоим ей значение true.**
2. **Напишем цикл, в котором счетчик пробегает диапазон от 2 до 100.**
3. **Написать второй цикл в котором идет перебор от 2 до значения счетчика первого цикла. Если остаток от деления равен 0, то isPrime = false и прерываем внутренний цикл.**
4. **В первом цикле проверяем условие isPrime, если ложно переходим к следующему шагу, в противном случае выводим, что число простое**

Область видимости переменной

1. Локальная–доступна внутри блока { } , в котором определена.

2. Глобальная–доступна в любой точке пространства имен (после определения).

Область видимости переменной

```
#include <iostream>
using namespace std;
int intGlobalVar;
char charGlobalVar;
double doubGlobalVar;

char charGlobalVar1 = 35;
char charGlobalVar2 = 'a';

int main()
{
    cout << "GlobalVars:" << endl;
    cout << "int intGlobalVar: " << intGlobalVar << endl;
    cout << "char charGlobalVar: " << charGlobalVar << endl;
    cout << "double doubGlobalVar: " << doubGlobalVar << endl;

    cout << "char charGlobalVar1: " << charGlobalVar1 << endl;
    cout << "char charGlobalVar2: " << charGlobalVar2 <<endl;


}
```

Область видимости переменной

GlobalVars:

int intGlobalVar: 0

char charGlobalVar:

double doubGlobalVar: 0

char charGlobalVar1: #

char charGlobalVar2: a

1. Неинициализированные глобальные переменные – значение 0

2. Неконстантные глобальные переменные – зло:

a) Значения могут быть изменены любой вызываемой функцией

b) Ухудшается масштабируемость (обычно предполагается 1 экземпляр)

Область видимости переменной

Будет работать не везде!

```
#include <iostream>
using namespace std;

int main()
{
    int localMainInt = 13;
    for (int i = 0; i < 3; ++i) {
        //переменная localMainInt видна здесь
        int localInt; //=1;
        cout << "localInt : " << localInt << endl;
        cout << "localMainInt : " << localMainInt << endl;
    }
    //cout<<localInt<<endl;// ошибка, переменная localInt не видна здесь!!!!!!!
    //а переменная localMainInt видна здесь
    return 0;
}
```

Область видимости переменной

1. **Значение не инициализированной локальной переменной – мусор!**
2. **Определяйте локальную переменную как можно ближе к месту ее использования!**
3. **Инициализируйте переменную сразу!**

Классы памяти

Классы памяти (время существования в памяти):

1. ***Автоматический***–время жизни совпадает со временем жизни функции, в которой определена.
2. ***Статический***–время жизни совпадает со временем жизни программы.

Глобальная переменная –статическая. Не может быть автоматической.

Локальная переменная –автоматическая. Может быть статической в случае использования ключевого слова **static**.

Классы памяти

```
#include <iostream>
using namespace std;

int main()
{
    for (int i = 0; i < 5; ++i) {
        static int a = 0;
        int b = 0;
        ++a;
        ++b;
        cout<< "static a: " << a <<“ ”;
        cout<< "nonstatic b: " << b << endl;
    }
   // std::cout << a << std::endl; // ошибка
переменная a не видна
}
```

Классы памяти

static a: 1 nonstatic b: 1

static a: 2 nonstatic b: 1

static a: 3 nonstatic b: 1

static a: 4 nonstatic b: 1

static a: 5 nonstatic b: 1

Функции

Функции – это блоки кода, выполняющие определенные операции. Если требуется, функция может определять входные параметры, позволяющие вызывающим объектам передавать ей аргументы. При необходимости функция также может возвращать значение как выходное.

z = f(x, y)

```
int getSum(int a, int b)
{
    int c = a + b;
    return c;
}
```

Объявление функции

```
тип_возвращаемого_значения имя функции(список аргументов) {
    //тело функции
    return возвращаемое значение;
}
```

```
int getSum(int a, int b)
{
    int c = a + b;
    return c;
}
```

```
int getSum(int a, int b)
{
    return a + b;
}
```

тип_возвращаемого_значения: void если ничего не возвращает

Пример использования функции

```
#include <iostream>

int getSum(int a, int b)
{
    return a+b;
}

int main()
{
    std::cout << getSum(5, 6) << std::endl;
}
```

Проблема

```
#include <iostream>

int main()
{
    std::cout << getSum(5, 6) << std::endl;
}

int getSum(int a, int b)
{
    return a+b;
}
```

Прототип функции

```
#include <iostream>

int getSum(int a, int b);// прототип функции

int main()
{
    std::cout << getSum(5, 6) << std::endl;
}

int getSum(int a, int b)
{
    return a+b;
}
```

Прототип функции

```
#include <iostream>

int getSum(int a, int b);// прототип функции

int main()
{
    std::cout << getSum(5, 6) << std::endl;
}

int getSum(int a, int b)
{
    return a+b;
}
```

Параметры по умолчанию

```
#include <iostream>

int getSum(int a = 2, int b = 3);// прототип
функции

int main()
{
    std::cout << getSum() << std::endl;
}

int getSum(int a, int b)
{
    return a+b;
}
```

Перегрузка функции

```
#include <iostream>

int getSum(int a = 2, int b = 3);// прототип функции
double getSum(double a = 2, double b = 3);// переопределение функции


int main()
{
    std::cout << getSum() << std::endl;
}

int getSum(int a, int b){return a+b;}
int getSum(double a, double b){return a+b;}
```

Реализация функций

Хорошо спроектированная функция имеет следующие характеристики:

•Полностью выполняет четко поставленную задачу;

•Не берет на себя слишком много работы;

•Не связана с другими функциями бесцельно;

•Помогает распознать и разделить структуру программы;

•Помогает избавиться от излишков, которые иначе присутствовали бы в программе.

Задание

1. Создать новый проект
2. Метод Лейбница по вычислению числа pi вынести в отдельную функцию, создать ее прототип
3. В качестве входного параметра функции задать число итераций в цикле
4. Число итераций считывается с консоли

Задание вычисление производной

1. Создать новый проект Derivative
2. Создать функцию, которая вычисляет производную, и ее прототип
3. Создать несколько математических функций (полином 2 степени, гауссову функцию, …)

Статические массивы

Массив — это последовательность объектов одного и того же типа, которые занимают смежную область памяти.

Пример

```
int a[4];
bool b[4];
for (int i = 0; i < 4; ++i) {
    cout << "address of a[" << i << "]: " << &a[i] << endl;
}

for (int i = 0; i < 4; ++i) {
    cout << "address of b[" << i << "]: " << &b[i] << endl;
}
```

Статические массивы

Int 4 байта

address of a[0]: 00000060E7EFF9A8
address of a[1]: 00000060E7EFF9AC
address of a[2]: 00000060E7EFF9B0
address of a[3]: 00000060E7EFF9B4

Bool 1 байта

address of b[0]: 00000060E7EFF9D4
address of b[1]: 00000060E7EFF9D5
address of b[2]: 00000060E7EFF9D6
address of b[3]: 00000060E7EFF9D7

Статические массивы

Массив — это последовательность объектов одного и того же типа, которые занимают смежную область памяти.

```
int a[4];// в элементах мусор
bool b[4];
cout <<"a = "<< a << endl;
cout << "b = " << b << endl;
```

a = 00000008904FF508

address of a[0]: 00000008904FF508

b = 00000008904FF534

address of b[0]: 00000008904FF534

Инициализация статические массивы

1. Нумерация массива с 0 !!!!!!! – для цикла for

```
int a[4] = { 1,2,3,4 };

int b[4];// поэлементная запись
b[0] = 1;
b[1] = 2;
b[2] = 3;
b[3] = 4;

int c[] = { 1,2,3,4,5,6,7 };//старая запись
int d[]{ 1,2,3,4,5,6,7 };// новая запись
int e[5]{}; // инициализируем 0
```

Инициализация статические массивы

```
const int size = 100; //без const недопустимо,

int newArray[size]{};
```