Объектно-ориентированное программирование

2017

Кто я?

Старший преподаватель кафедры 806

Дзюба Дмитрий Владимирович

ddzuba@yandex.ru

Примеры https://github.com/DVDemon/mai_oop

канал телеграмм https://t.me/oop_mai

Базовые требования к слушателям

1. Знание языка программирования C
   при изложении материала будем считать, что слушатель знает основные конструкции языка C, типы данных и правила написания программ

2. Знание операционной системы Microsoft Windows 7/8/10
   практические занятия будут проходить на компьютерах, работающих под управлением Microsoft Windows 7

3. Знание среды разработки Microsoft Visual Studio 2013/2015
   лабораторные работы должны делаться в Microsoft Visual Studio, мы будем создавать консольные приложения с unmanaged кодом

Отчетность по курсу **рейтинг**

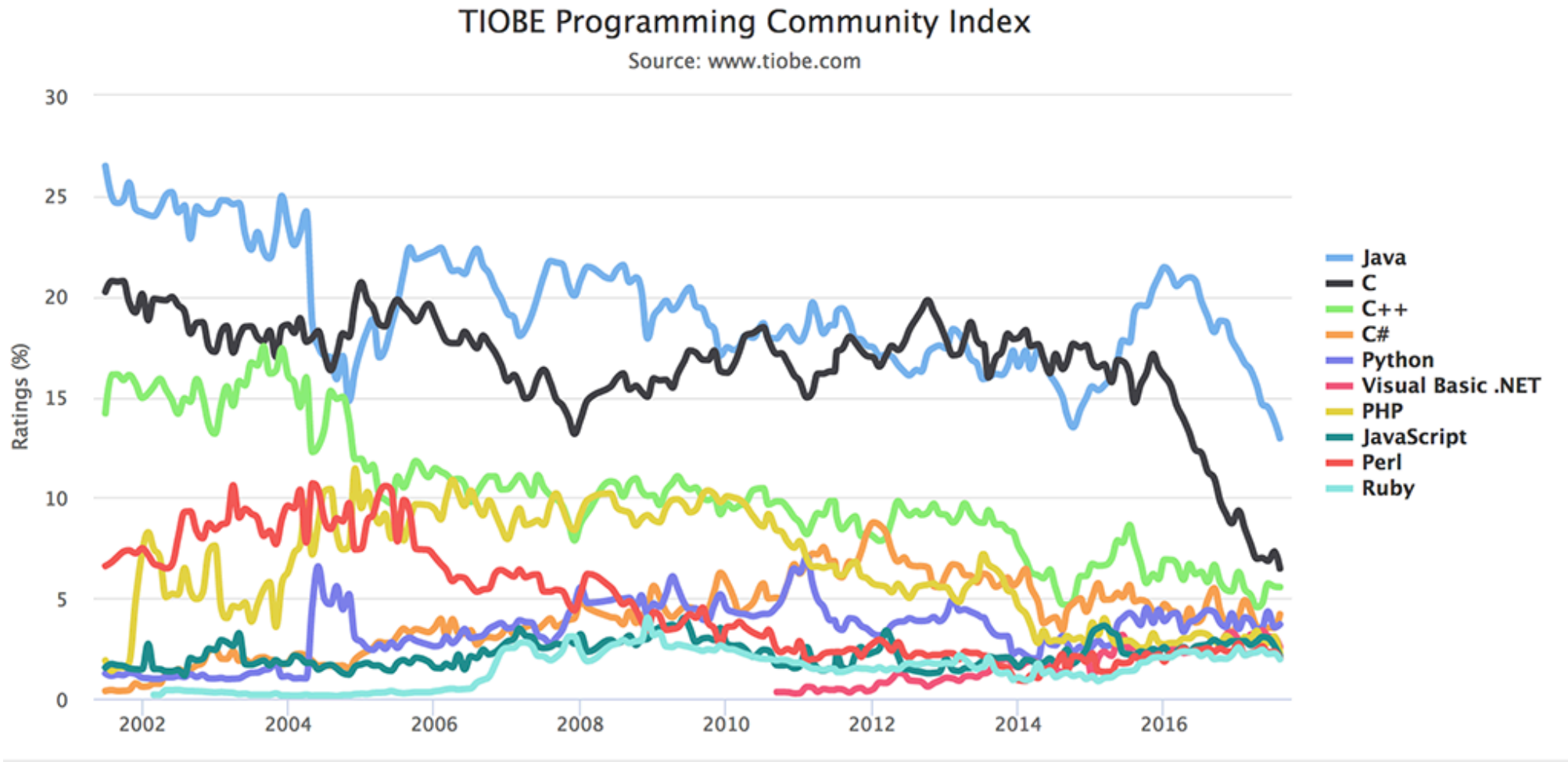
5-бальная система	Рейтинговая система	Европейская система
5 - Отлично	**90-100**	A
4 – Хорошо	**82-89**	B
	75-81	C
3 - Удовлетворительно	**67-74**	D
	60-66	E
2 - Неудовлетворительно	**Менее 60**	F

Балы даются:

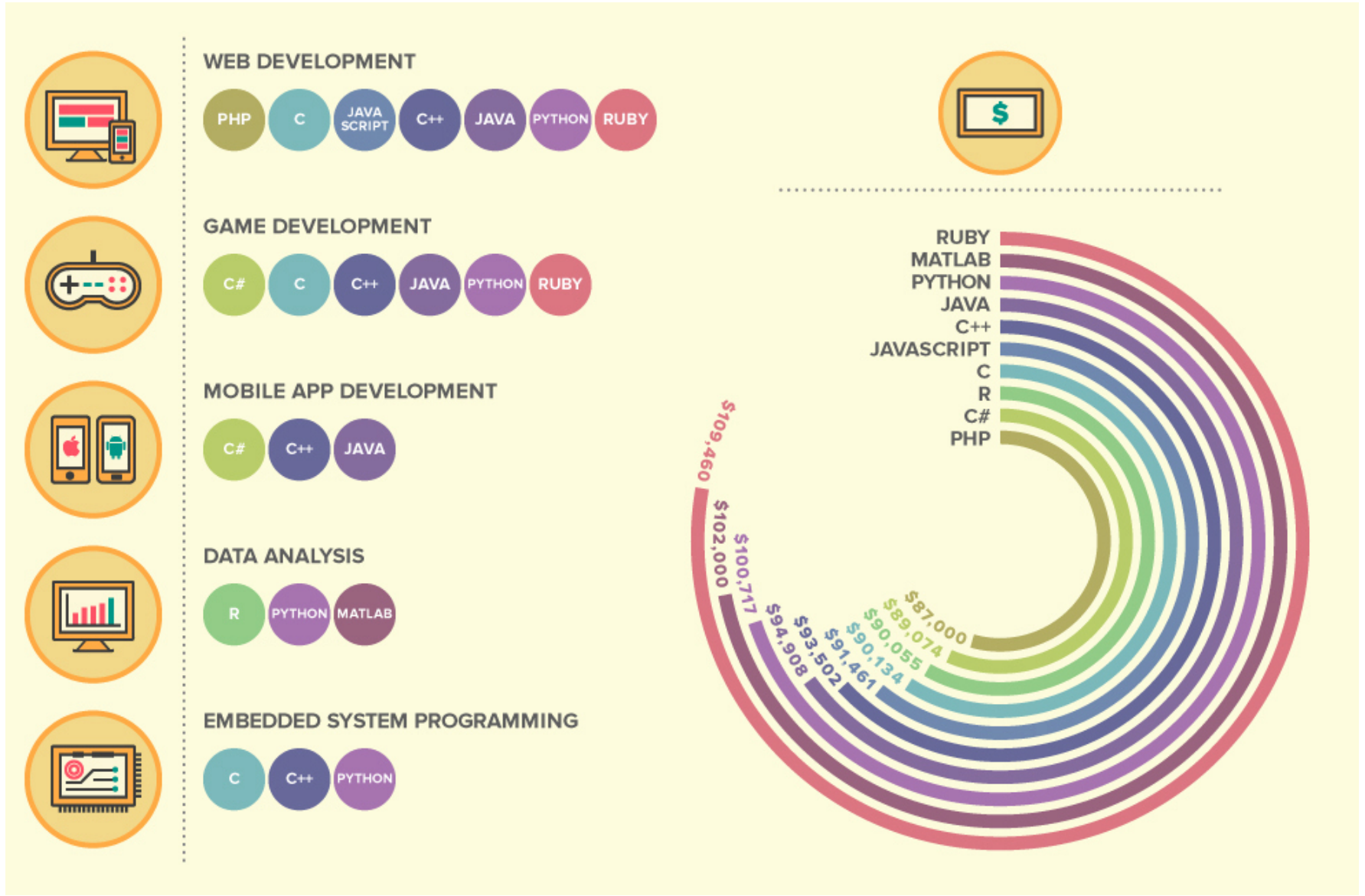
1. Вовремя сделанная и сданная Лабораторная работа (9 шт) – 10 баллов.
2. Лабораторная работа сданная с задержкой в две недели и более – 5 баллов.
3. Зачет (два задания) по 15 балов за задание (итого до 30).

Расписание занятий

1. Лекции проходят каждую неделю по понедельникам с 16.00. В 444 ГУК.
2. Лабораторные работы проходят раз в две недели по пятницам с 8.30 до 15.00 (по две группы) в 440Б.



В курсе мы будем изучать язык программирования C++ ориентируясь на 11/14 стандарт.


WEB DEVELOPMENT
PHP
C
JAVA SCRIPT
C++
JAVA
PYTHON
RUBY
GAME DEVELOPMENT
C#
C
C++
JAVA
PYTHON
RUBY
MOBILE APP DEVELOPMENT
C#
C++
JAVA
DATA ANALYSIS
R
PYTHON
MATLAB
EMBEDDED SYSTEM PROGRAMMING
C
C++
PYTHON
$
RUBY
MATLAB
PYTHON
JAVA
C++
JAVASCRIPT
C
R
C#
PHP
$109,460
$102,000
$100,717
$94,908
$93,502
$91,461
$90,134
$90,055
$89,074
$87,000

Лабораторные работы

№	Цель
1	• Изучение базовых понятий ООП. • Знакомство с классами в C++. • Знакомство с операциями ввода-вывода из стандартных библиотек.
2	• Закрепление навыков работы с классами. • Знакомство с перегрузкой операторов. • Знакомство с дружественными функциями. • Создание простых динамических структур данных. • Работа с объектами, передаваемыми «по значению».
3	• Закрепление навыков работы с классами. • Знакомство с умными указателями.
4	• Знакомство с шаблонами классов. • Построение шаблонов динамических структур данных.
5	• Закрепление навыков работы с шаблонами классов. • Построение итераторов для динамических структур данных.
6	• Закрепление навыков по работе с памятью в C++. • Создание аллокаторов памяти для динамических структур данных.
7	• Создание сложных динамических структур данных. • Закрепление принципа OCP.
8	• Знакомство с параллельным программированием в C++.
9	• Знакомство с лямбда-выражениями.

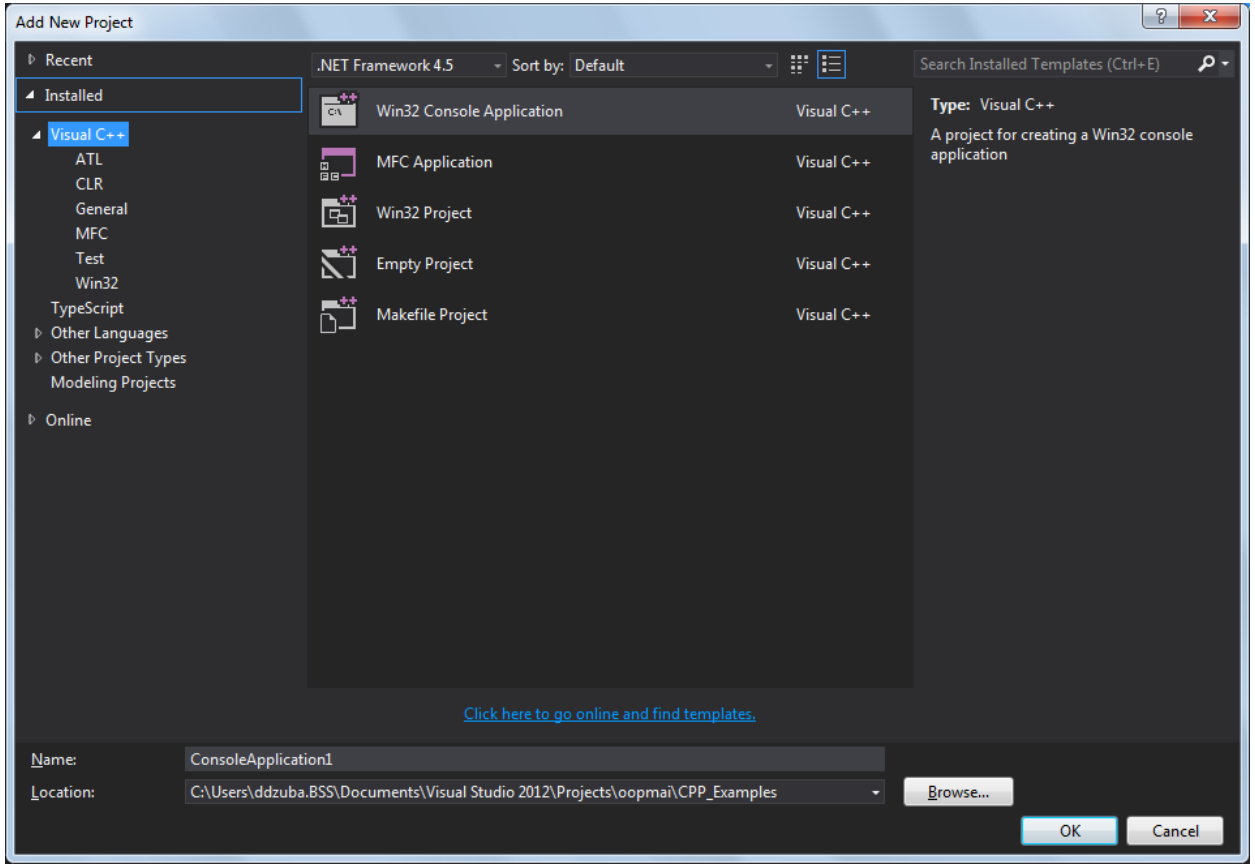
Среда разработки

Допускается использование следующих сред разработки/компиляторов:

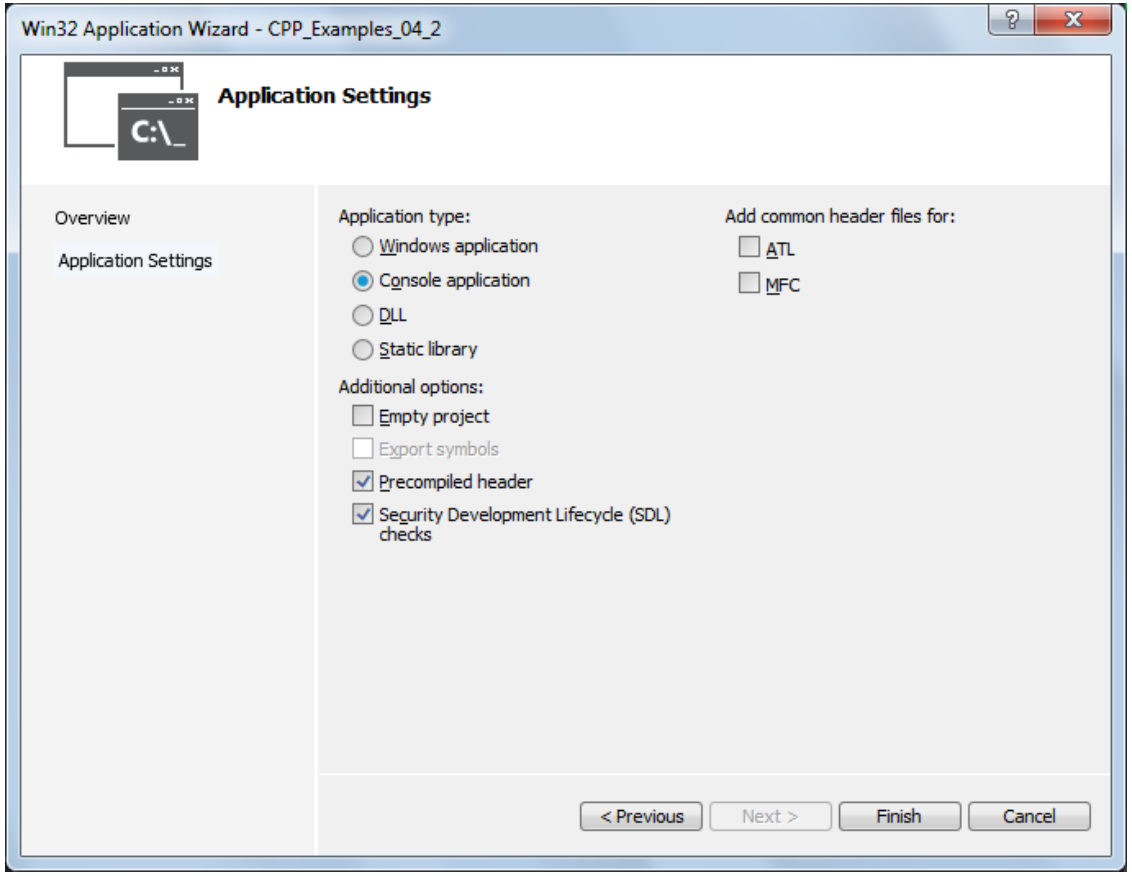
- Microsoft Visual Studio 2013 для MS Windows 7/8.1/10
- X-Code (clang) для MacOS X 10.x
- gcc для Linux (например, Ubunta).

Допускается использование других компиляторов C++ поддерживающих стандарт C++ 11 и выше.

Создание проекта на C++ в Microsoft Visual Studio 2013 [1/2]



Создание проекта на C++ в Microsoft Visual Studio 2013 [2/2]



Общие понятия

ЛЕКЦИЯ №1

Семейство языков C

C

1972 , Dennis Ritchie @ Bell Labs.

Императивный язык программирования

C++

1979 , Bjarne Stroustrup @ Bell Labs.

Императивный, объектно-ориентированный язык программирования

Язык C

1. Компилируемый.
2. Императивный.
3. Ручное управление памятью.
4. Используется когда нужно написать программу, которая:
   - Эффективна по скорости работы.
   - Эффективна по потребляемой памяти.

Вспоминаем C

Вспоминаем С
локальные и глобальные переменные

```
int x;
int y, z;
x= 1;
/* У функции могут быть локальные переменные    */
void foo() {
int x;
x= 2;
}
/* Параметры имеют локальную область видимости */
void bar(int x) {
x= 3;
}
```

Вспоминаем С
условия

```
int foo( int x) {
/* Используются обычные булевы операторы . */
if (3 ==x) {
return 0;
}}
/* Условия используют целый тип, 1 – это истина. */
int bar()
{
if     (1) {return    0; }
}
```

Вспоминаем С

ЦИКЛЫ

```
void foo() {/*Цикл в стиле for*/
int i;
for (i=1; i < 10; ++i ) {
printf (”%d\n”, i);}}

void bar() {/* Цикл в стиле while*/
int lcv =0;
while (lcv < 10) {
printf (”%d\n”, lcv++);
}}
```

Вспоминаем С
ВЫЗОВЫ

```
/* Declaration . */
void print sum(int , int );
/* Each executable needs to have a main function with type int . */
int main() {
  print_sum(3, 4);
  return 0;
}
/* Definition . */
void print_sum( int arg1 , int arg2)
{/* Body defined here . */ }
```

Вспоминаем С
модули

1. Файлы определений Header (обычно имеют расширение .h).
   Говорят компилятору, что функции где-то определены.
2. Файлы с описанием функций (обычно имеют расширение .c).
   Содержат текст описания алгоритмов функций.
3. Возможность «включить» Header в файл с описанием функций:
   - #include <stdio.h> //подключаем файл из стандартной библиотеки
   - #include “myfile.h” //подключаем локальный файл

Итого: C

1. Мы можем писать большие алгоритмы.
2. Мы можем организовывать программный код в большое количество библиотек и модулей.

О работе с памятью в C

Ручное управление памятью в C

Цели процесса

Позволить программе помечать области памяти как «занятые» полезной информацией.

Позволить программы помечать области памяти как «не занятые» по окончании работы. Что бы эти области могли использовать другие алгоритмы и программы.

Что есть в C

В библиотеке stdlib.h есть функции malloc и free.

Управление памятью: Куча (heap)

1. Куча – это область памяти, который может использовать программа.
2. Кучу можно сравнить с гигантским массивом.
3. Для работы с кучей используется специальный синтаксис указателей.
4. Вся программа может получить доступ к куче.

Addr.	Contents
⋮	⋮
0xbee	0xbeef
0xbf4	0xfeed
⋮	⋮

Пример работы с кучей в C

Example01_MemoryC

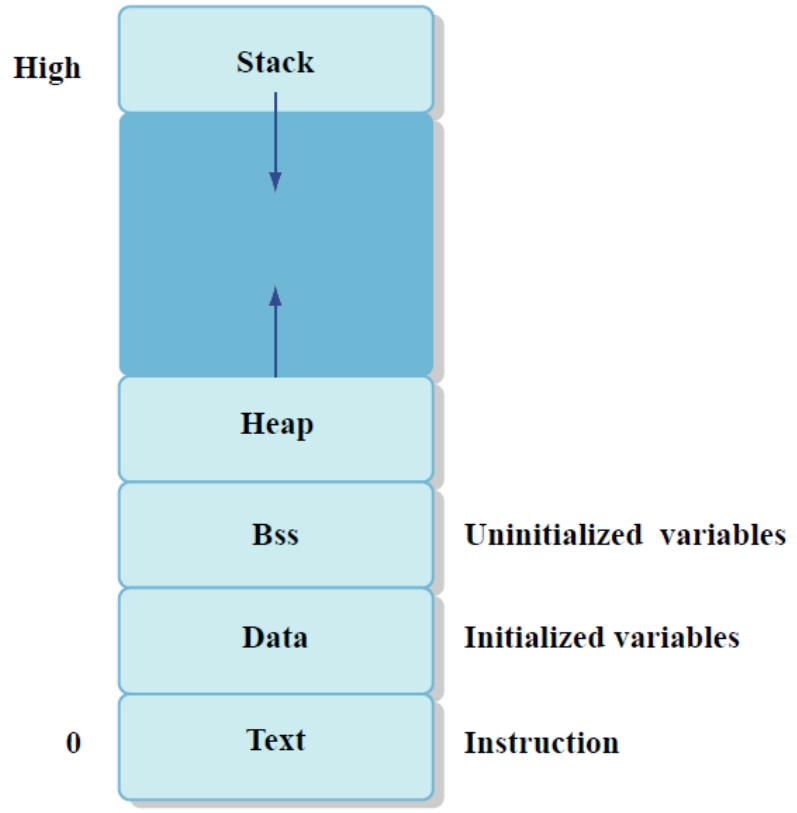
```
#include "stdlib.h" // работа с памятью
#include "stdio.h" // работа с вводом и выводом
#include "string.h" // работа со строками

int _tmain(int argc, _TCHAR* argv[])
{
// выделяем память
char *pointer = (char*) malloc(sizeof(char)*100);
// копируем в память данные
strcpy(pointer,"Hello World!");
// получаем данные из памяти
printf("%s",pointer);
// освобождаем указатель
free(pointer);
// ждем нажатия любой клавиши
getchar();
return 0;
}
```

Выделение памяти в стеке (stack)

Функции С размещаются в стеке:

1. Функции помещаются в стек, в момент вызова.
2. Функции удаляются из стека в момент когда вызывается return.
3. Функция может использовать любую память в пределах стека.



Переменные «на стеке»

Example02_StackFault

```
1. #include "stdlib.h" // работа с памятью
2. #include "stdio.h" // работа с вводом и выводом
3. #include "string.h" // работа со строками
4. int a[10];
5. int *array_func(int val) {
6. //    int a[10];
7. for (int i = 0; i < 10; i++) a[i] = val;
8. return a;
9. }
10.int main(int argc, char** argv) {
11.int *array = array_func(3);
12.for (int i = 0; i < 10; i++)        printf("Array is %d\n", *(array+i));
13.return 0;
14.}
```

Основные понятия ООП

Объект

«Объект представляет собой конкретный опознаваемый предмет, единицу или сущность (реальную или абстрактную), имеющую четко определенное функциональное назначение в данной предметной области»

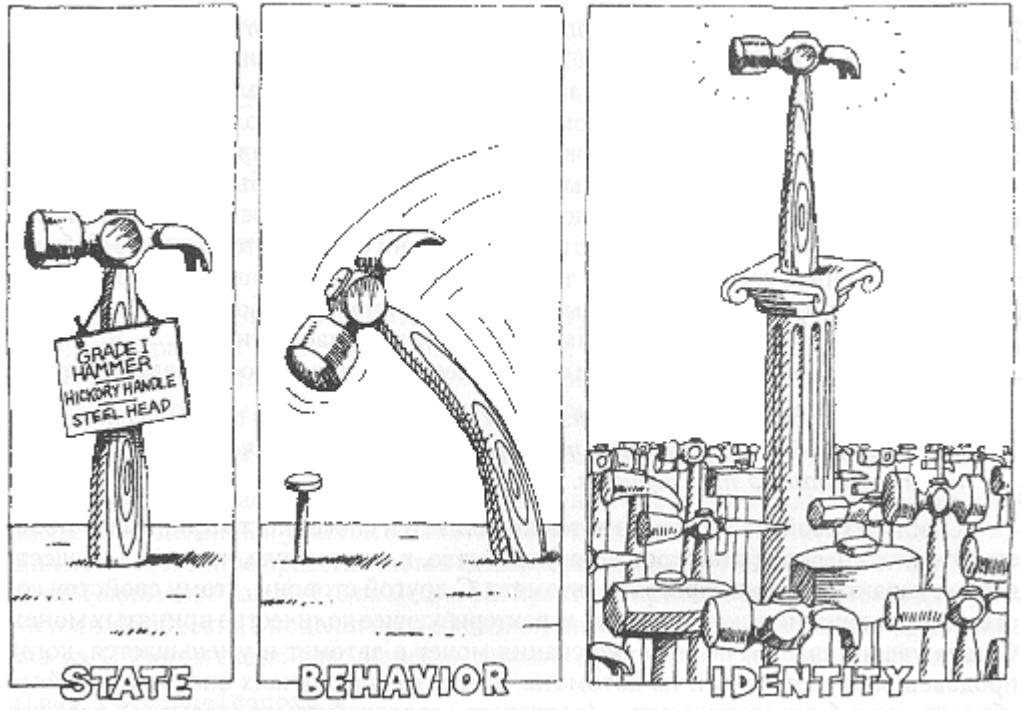
Smith, M. and Tockey, S. 1988. An Integrated Approach to Software Requirements Definition Using Objects. Seattle, WA: Boeing Commercial Airplane Support Division, p.132.

Свойства объекта

1. Состояние
   в любой момент времени объект находится в каком-либо состоянии, которое можно измерить / сравнить / скопировать

2. Поведение
   объект может реагировать на внешние события либо меняя свое состояние, либо создавая новые события

3. Идентификация
   объект всегда можно отличить от другого объекта



Класс

1. **Определение.**
   Классом будем называть группу объектов, с общей структурой и поведением.
2. Смысл программы на C++ это описание классов!
3. Даже если нужен всего один объект – мы будем описывать класс.

Очень простой класс объектов

Example03_FirstClass

```
class MyClass
{
public:
    int Number;
    void doSomething();
};
```

class – ключевое слово

public – область видимости атрибутов и методов класса

`int Number` – атрибут класса

`void doSomething()` - метод класса

Как работать с вводом/выводом в C++?

http://www.cplusplus.com/reference/iostream/

Механизм для ввода-вывода в Си++ называется потоком . Название произошло от того, что информация вводится и выводится в виде потока байтов – символ за символом.

- Класс **istream** реализует поток ввода,
- Класс **ostream** – поток вывода.

Библиотека потоков ввода-вывода определяет три глобальных объекта: cout, cin и cerr.

- **cout** называется стандартным выводом,
- **cin** – стандартным вводом,
- **cerr** – стандартным потоком сообщений об ошибках.

cout и **cerr** выводят на терминал и принадлежат к классу **ostream**, **cin** имеет тип **istream** и вводит с терминала. Разница между **cout** и **cerr** существенна в **Unix** – они используют разные дескрипторы для вывода. В других системах они существуют больше для совместимости.

Вывод осуществляется с помощью операции **<<,** ввод с помощью операции **>>.**

```
int x; cin >> x; // ввод числа X
```

Пример по работе с потоками

Example04_Stream

```
1.#include <cstdlib>
2.#include <iostream>
3.#include <string>
4.#include <fstream>
5.int main(int argc, char** argv) {
6.    std::string file_name;
7.    std::string file_text;
8.    std::cout << "Please enter file name:";
9.    std::cin >> file_name;
10.    std::ofstream out_file(file_name, std::ofstream::out);
11.    std::cout << "Please enter file text:";
12.    while (std::cin >> file_text) out_file << file_text << std::endl;
13.    out_file.close();
14.    std::cout << "Result:" << std::endl;
15.    std::ifstream in_file(file_name);
16.    while (in_file >> file_text) std::cout << file_text << std::endl;
17.    in_file.close();
18.    return 0;
19.}
```

Перегрузка операций

Почему операция **std::cin >> file_text** имеет смысл?

В C++ существуют механизмы, которые позволяют сопоставлять арифметический и другие операции, такие как побитовый сдвиг обычным функциям!

Это позволяет лучше описывать типы. Мы можем описать не просто класс, но и операции с объектами этого класса.

Как это работает мы узнаем немного позже.

Namespace

Example05_Namespace

```
1.#include <cstdlib>
2.#include <iostream>

3.namespace MyNameSpace{
4.    int value = 0;
5.}

6.int value = 0;
7.int main(int argc, char** argv) {
8.    MyNameSpace::value = 7;
9.    std::cout << value << std::endl;
10.   std::cout << MyNameSpace::value << std::endl;
11.
12.   using namespace MyNameSpace;
13.   //std::cout << value << std::endl;
14.   return 0;
15.}
```

Спасибо!

НА СЕГОДНЯ ВСЕ