Научно-исследовательский практикум

СТРУКТУРЫ. КЛАССЫ.

Перечисления (enum)

```cpp
#include <iostream>

using namespace std;

enum Colors {Red, Green, Blue=10, Black};

int main(int argc, char *argv[])
{
        cout << "Red=" << Red << endl;
        cout << "Green=" << Green << endl;
        cout << "Blue=" << Blue << endl;
        cout << "Black=" << Black << endl;
        int colorNum = Green;
        Colors color = Red;
        // Colors color2 = 10; // Ошибка
        return 0;
}
```

Трудности с множеством переменных – variables_project

```
int main()
{
    //  Например, у нас есть навигационная система, выдающая следующие данные:
    //  абсолютную скорость, компоненты скорости в связных координатах Vx и Vz
    //  курсовой угол, крен и дифферент
    //  а так же дату и точное время (до 1 секунды)
    //  Которые мы будем фильтровать
    double vRecv, vXRecv, vZRecv, headRecv, pitchRecv, rollRecv;
    char year, month, day, hour, minute, second;

    initialize();

    while (true)
    {
        recvData(vRecv, vXRecv, vZRecv, headRecv, pitchRecv, rollRecv, year, month, day, hour, minute, second);
        filterData(vRecv, vXRecv, vZRecv, headRecv, pitchRecv, rollRecv);
        doSomething(vRecv, vXRecv, vZRecv, headRecv, pitchRecv, rollRecv, year, month, day, hour, minute, second);
    }


    return 0;
}
```

Трудности с множеством переменных

Использование глобальных переменных

```
#include <iostream>
#include <ctime>        //  Для реализации таймера и rand()
#include <cmath>
#include <unistd.h>     //  Для функции usleep
#include <stdlib.h>     //  Для функции system

using std::cout;
using std::endl;

const int filterSize = 10;

double gV[filterSize], gVx[filterSize], gVz[filterSize],
       gHeading[filterSize], gRoll[filterSize], gPitch[filterSize];

int initialize();

int recvData(double& v, double& vx, double& vz, double& heading, double& pitch, double& roll,\
             char& year, char& month, char& day, char& hour, char& minute, char& second);

int filterData(double& v, double& vx, double& vz, double& heading, double& pitch, double& roll);

int doSomething(double& v, double& vx, double& vz, double& heading, double& pitch, double& roll,\
                char& year, char& month, char& day, char& hour, char& minute, char& second);

int main()
{
    //  Например, у нас есть навигационная система, выдающая следующие данные:
    double  vRecv,      //  Абсолютная скорость
            vXRecv,     //  Маршевая компонента скорости
            vZRecv,     //  Лаговая компонента скорости
            headRecv,   //  Угол курса
            pitchRecv,  //  Угол крена
            rollRecv;   //  Угол дифферента
    char    year,       //  Год
            month,      //  месяц года
            day,        //  день месяца
            hour,       //  час
            minute,     //  минута
            second;     //  секунда

    initialize();
```

Пользовательские типы данных

В УЗКОМ СМЫСЛЕ

Класс в *узком* смысле— одноименный составной пользовательский тип данных, являющийся контейнером для данных и алгоритмов их обработки. Вводится в текст программы определением типа со спецификатором class;

В ШИРОКОМ СМЫСЛЕ

Класс в *широком* смысле— любой составной пользовательский тип данных, агрегирующий данные и алгоритмы их обработки. Вводится в текст программы определением типа с одним из спецификаторов struct, union или class.

Структуры – чуть проще

```
#include <iostream>
#include <ctime>
#include <cmath>
#include <unistd.h>     //  Для функции usleep
#include <stdlib.h>    //  Для функции system

using std::cout;
using std::endl;

const int filterSize=10;

struct NavSystem
{
    double v, vx, vz,
           head, roll, pitch;
    char year, month, day,
         hour, min, sec;
};

NavSystem latest[filterSize];

int initialize();
int recvData(NavSystem& toRecv);
int filterData(NavSystem& toRecv);
int doSomething(NavSystem& toRecv);

int main()
{
    NavSystem data;

    initialize();

    while (true)
    {
        recvData(data);
        filterData(data);
        doSomething(data);
    }
    return 0;
}
```

Структуры – изменения

```
int initialize()
{
    for (int i=0; i<filterSize; i++)
    {
        latest[i].v=0;
        latest[i].vx=0;
        latest[i].vz=0;
        latest[i].head=0;
        latest[i].roll=0;
        latest[i].pitch=0;
    }
    return 0;   //  тут, в общем, нечему ломаться
}
```

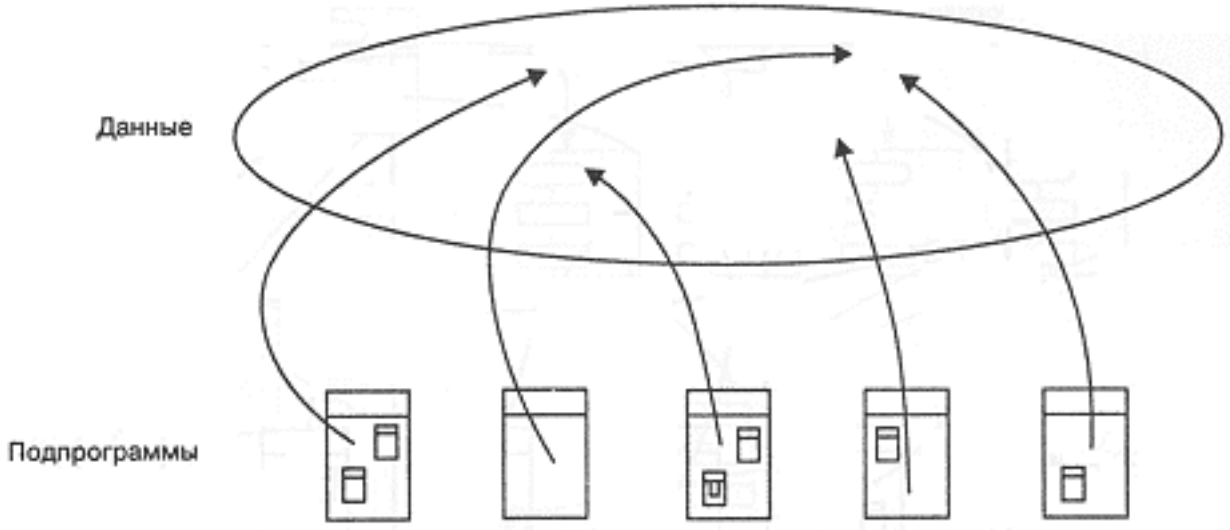
```
int filterData(NavSystem& recvd)
{
    for (int i=0; i<filterSize-1; i++)
    {
        latest[i] = latest[i+1];
    }
    latest[filterSize-1] = recvd;
    for (int i=0; i<filterSize-1; i++)
    {
        recvd.v +=latest[i].v;
        recvd.vx+=latest[i].vx;
        recvd.vz+=latest[i].vz;
        recvd.head+=latest[i].head;
        recvd.roll+=latest[i].roll;
        recvd.pitch+=latest[i].pitch;
    }
    recvd.v /= filterSize;
    recvd.vx /= filterSize;
    recvd.vz /= filterSize;
    recvd.head /= filterSize;
    recvd.roll /= filterSize;
    recvd.pitch /= filterSize;

    return 0;
}
```

Классы

До этого:

- Переменные с различной областью видимости
  - Локальные переменные, видимые внутри функции
  - Глобальные переменные, видимые везде
- Функции, принимающие значения



Классы

У класса есть собственные «внутренние» переменные

У класса есть собственные, связанные только с ним, функции

Функция класса имеет доступ к его внутренним переменным

Классы

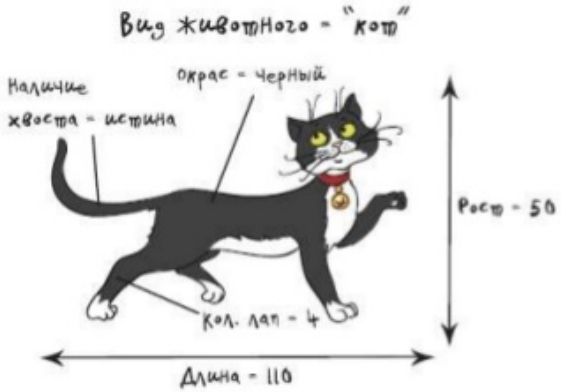
Класс Animal

Animal Barsik;

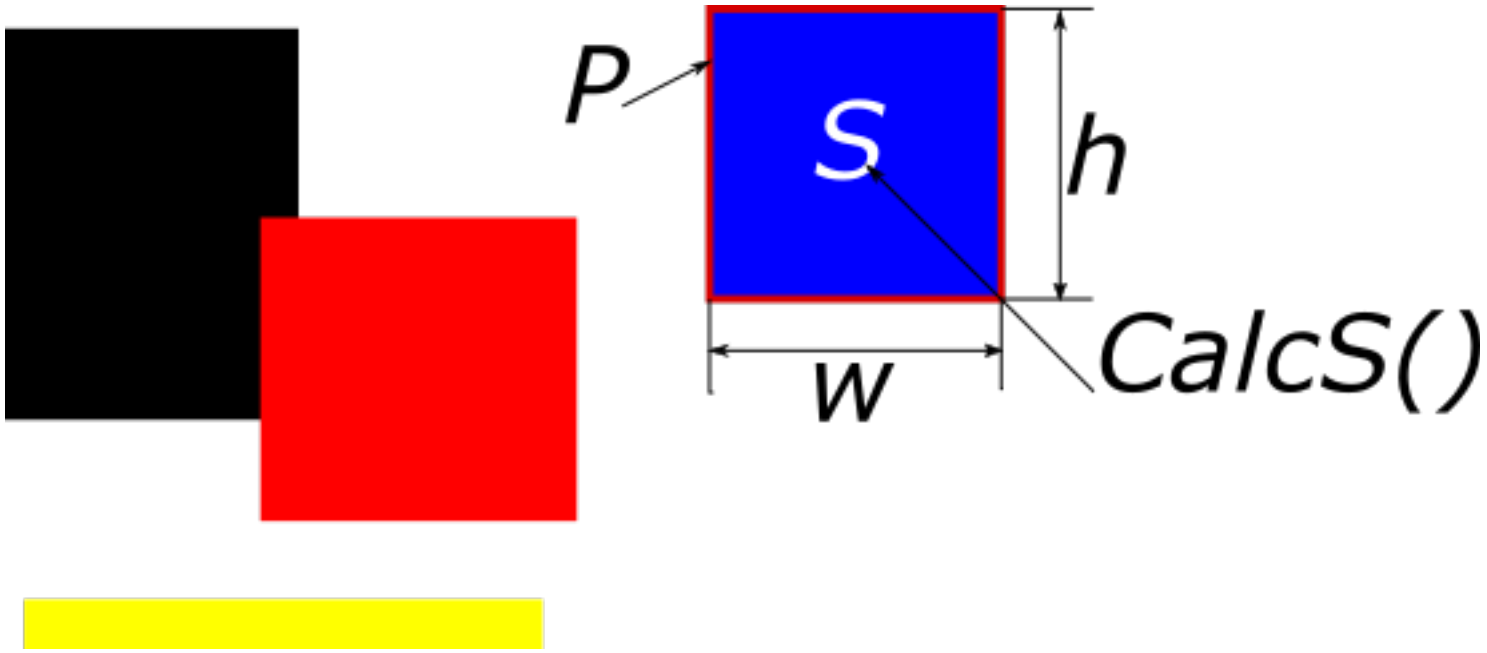
Kind of animal = "Cat" ("Кот").
Height = 50 cm (большой котяра!).
Length = 110 cm (это рысь скорее, а не домашняя кошка!).
Number of legs = 4.
Color = "Black" (черный).
Has *tail* = true (истина).
Is mammal = true (истина).

```
class Animal
{

}
```



Прямоугольники



Прямоугольники

```
struct Rectangle
{
        // Методы
        int CalcS() {return m_Height*m_Width;} // Площадь
        int CalcP() {return 2*(m_Height+m_Width);} //
Периметр
        void Draw(char filler = '*')
        {
                for (int i=0; i<m_Height; i++)
                {
                        for (int j = 0; j<m_Width; j++)
                                cout << filler;
                        cout << endl;
                }
        }

        // Переменные
        int m_Height; // Ширина
        int m_Width; // Высота
};
```

Прямоугольники

```
int main(int argc, char *argv[])
{
        Rectangle a;
        a.m_Height = 2;
        a.m_Width = 12;
        cout << "Area of a=" << a.CalcS() << " and preimetr " << a.CalcP() << endl;
        a.Draw();
        Rectangle b;
        b.m_Height = 3;
        b.m_Width = 8;
        cout << "Area of b=" << b.CalcS() << " and preimetr " << b.CalcP() << endl;
        b.Draw('+');
}
```

Прямоугольники

```
Starting /Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj...
Area of a=24 and preimetr28
**********
**********
Area of b=24 and preimetr22
++++++++
++++++++
++++++++
/Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj exited with code 0
```

Конструктор и деструктор

Конструктор – специальная функция, автоматически вызываемая при создании класса

Имя конструктора всегда совпадает с именем класса

Конструктор отвечает за инициализацию экземпляра

Вызывать конструктор вручную НЕ НУЖНО!

Конструктор и деструктор

```
struct Rectangle
{
        // Конструктор
        Rectangle()
        {
                cout << "Rectangle constructor
called..." << endl;
                m_Height = 1.0; m_Width = 1.0;
        }
        // Методы
        ...
};

int main(int argc, char *argv[])
{
        int c(5), b(3);
        cout << "A=" << c << " and b=" << b << endl;
        Rectangle a;
        cout << "Size of rectangle is " << a.m_Height
<< "x" << a.m_Width << endl;
}
```

Конструктор и деструктор

```
// Конструктор
Rectangle()
{
        cout << "Rectangle constructor called..." <<
endl;
        m_Height = 1.0; m_Width = 1.0;
}
Rectangle(int height, int width)
{
        cout << "Rectangle constructor called..." <<
endl;
        m_Height = height;
        m_Width = width;
};

//////////////////////////////////////////

int main(int argc, char *argv[])
{
        Rectangle a(3,5);
        cout << "Size of rectangle is " << a.m_Height
<< "x" << a.m_Width << endl;
        a.Draw();
}
```

Деструктор

Конструктор – специальная функция, автоматически вызываемая при удалении объекта класса

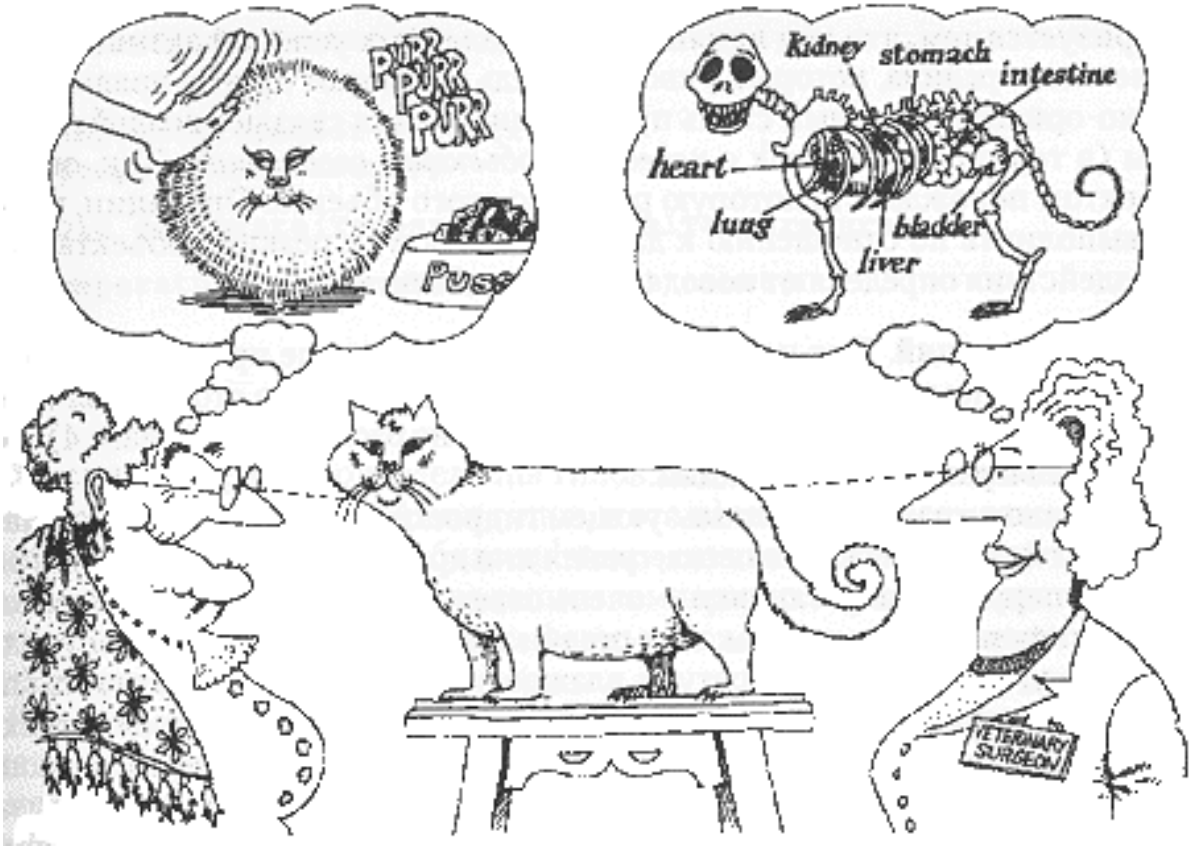
Имя конструктора всегда совпадает с именем класса ~Classname

Деструктор никогда не вызывается!

```
~Rectangle()
{
        cout << "Rectangle destructor called..." << endl;
}
```

Инкапсуляция

Мы будем пытаться
«скрыть»
ненужные данные



Инкапсуляция

Используются ключевые слова

- Public – данные, видимые снаружи класса
- Private – данные, видимые только изнутри

```
// Переменные
private:
        int m_Height; // Ширина
        int m_Width; // Высота
```

Мы должны получить ошибку компиляции на предыдущем коде

Различия структур и классов

STRUCT

Все переменные-члены и функции по умолчанию являются public

Если мы хотим сделать их private, мы должны вручную указать спецификатор

CLASS

Все переменные-члены и функции по умолчанию являются private

Если мы хотим сделать их public, мы должны вручную указать спецификатор

Сеттеры и геттеры

Если переменная закрыта, до нужны методы для работы с ней

Геттер – метод для получения переменной

Сеттер – для её изменения

```
int GetH() const
{
        return m_Height;
}

void SetH(int height)
{

}
```

Класс

```
struct Rectangle
{
public:
        // Конструктор
        Rectangle()
        {
                cout << "Rectangle constructor
called..." << endl;
                m_Height = 1.0; m_Width = 1.0;
        }
        // Методы

private:

        int m_Height; // Ширина

        int m_Width; // Высота
};

int main(int argc, char *argv[])
{
        int c(5), b(3);
        cout << "A=" << c << " and b=" << b << endl;
        Rectangle a;
        cout << "Size of rectangle is " << a.m_Height
<< "x" << a.m_Width << endl;
}
```

Что не так?

Программа из предыдущего примера не будет компилироваться

Что в ней надо изменить?