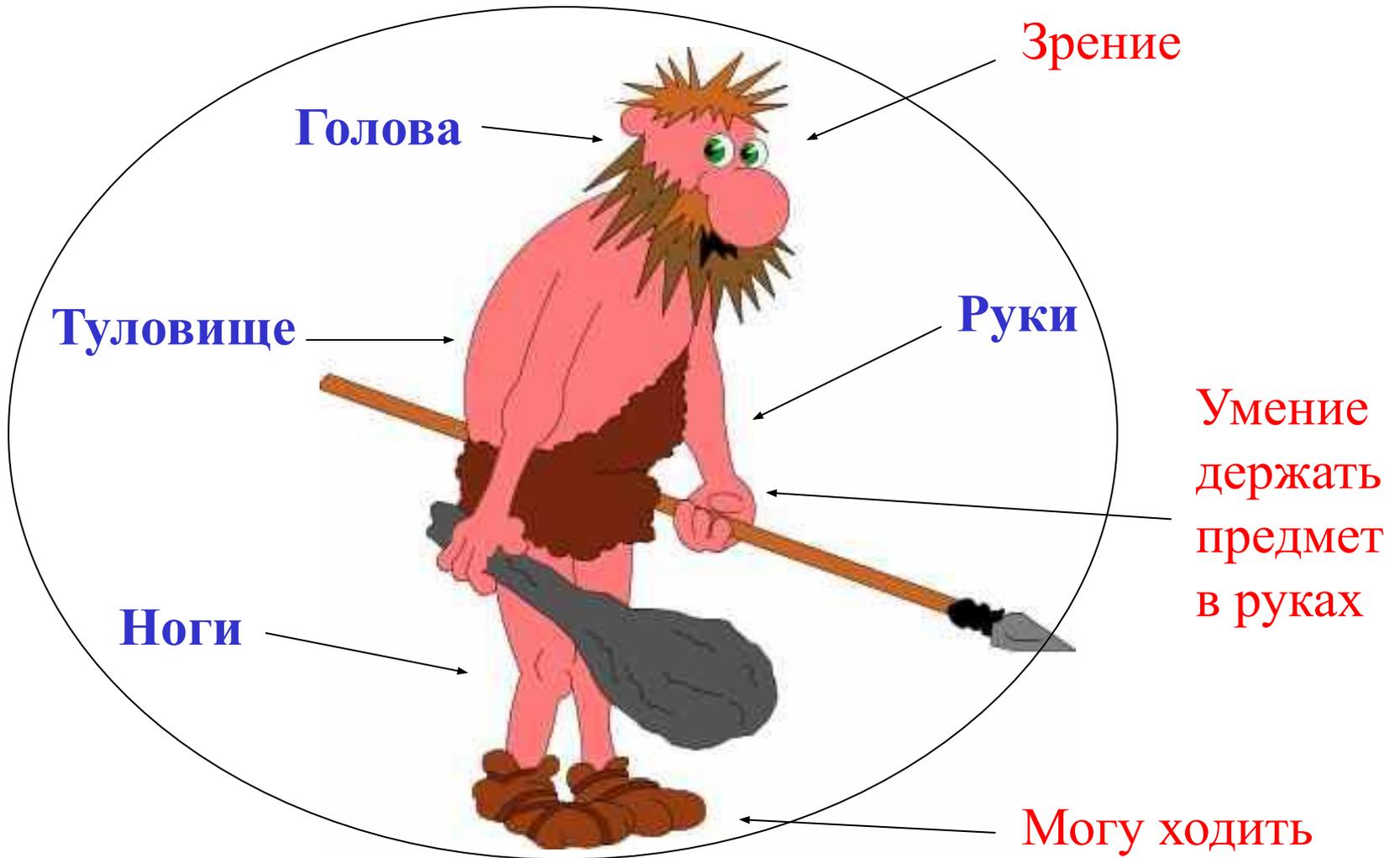
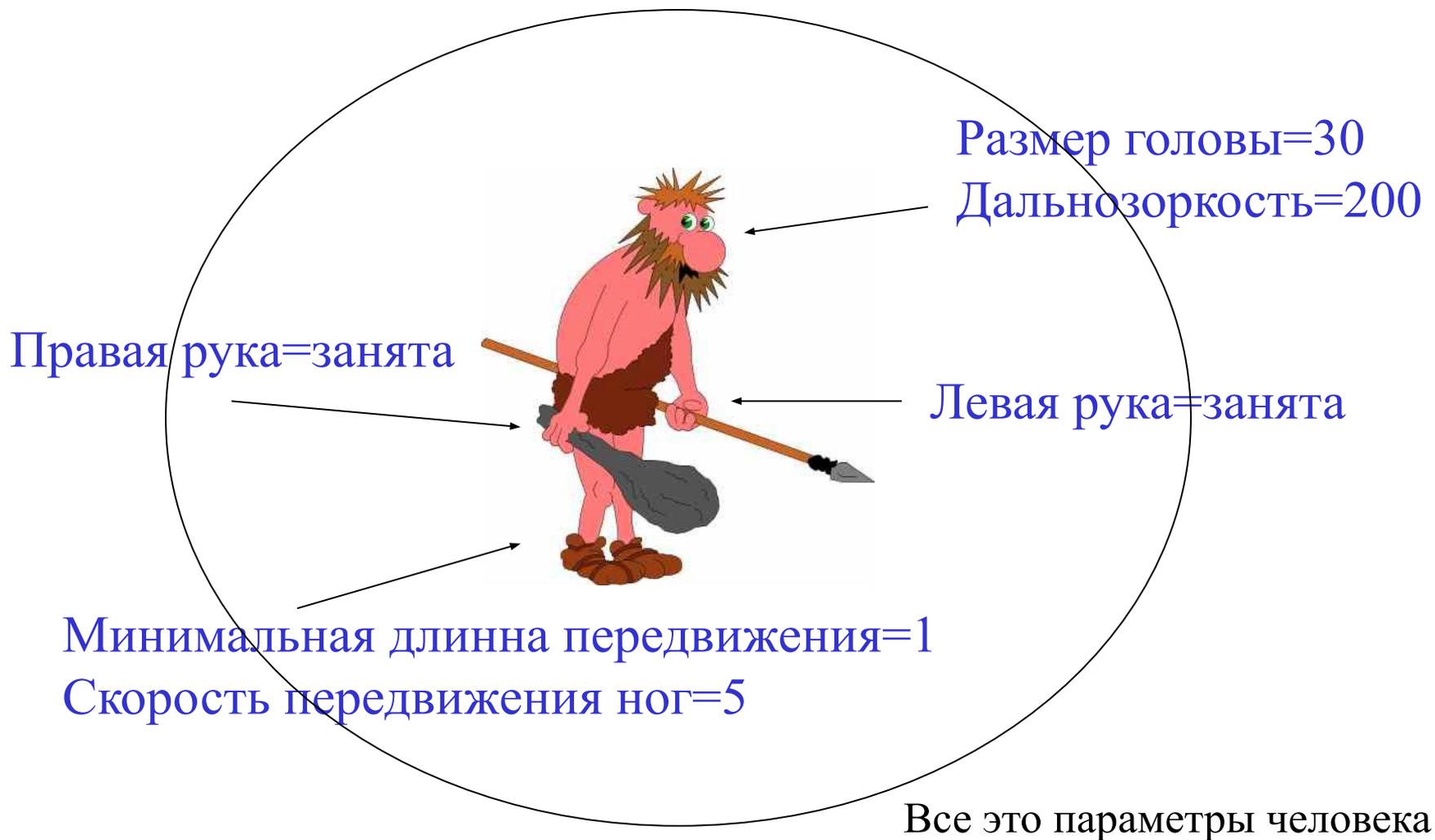


Visual C++

Microsoft®
**Visual
Studio**







Учитывая дальность зрения можно определить область плоскости, которую человек видит и куда может пойти.



Параметры человека:

Позиция_x=3;
Позиция_y=4;
Двигается=false;
Сила_правой_руки=10;
Сила_левой_руки=6;
Правая_рука_занята=true;
Левая_рука_занята=true;
Дальнозоркость=200;
Быстрота_шага=5;
Жизненная_сила=100;
Интеллект=0;

Функциональные возможности:

Осмотреться();
Позиция_свободна(x,y);
Передвинуться_на_позицию(x,y)
Идти_вперед();
Идти_назад();
Идти_влево();
Идти_вправо();
Остановиться();
Проверить_заняты_ли_руки();
Взять_предмет(какой);
Положить_предмет(какой);
и т.д.

Одни только свойства и навыки не могут наделить нашего человека интеллектом, поскольку он должен научиться ими управлять. Управлять ими он будет с помощью функций!

```
Boolean Идти_вперед();
{
if (Жизненная_сила!=0)
{
    if (Позиция_свободна(x,y)=true)
        {
            x=x+Быстрота_шага;
            return true;
        }
    else
        {
            Остановить_я();
            return false;
        }
}
}
```

Таким образом, совмещение в одном объекте как свойств, так и действий над ними является базовым принципом объектно-ориентированного программирования (ООП) и называется ..

ИНКАПСУЛЯЦИЯ

Полиморфизм - свойство, которое позволяет одно и тоже имя использовать для решения двух или более схожих, но технически разных задач

Наследование - это процесс, посредством которого один объект может приобретать свойства другого

Класс объявляется с помощью ключевого слова `class`

```
class имя_класса {
```

закрытые функции и переменные класса

```
public:
```

открытые функции и переменные класса

```
} список_объектов;
```

Объявление класса MAN не задает ни одного объекта типа MAN, оно определяет только тип объекта, который будет создан при его фактическом объявлении. Чтобы создать объект, необходимо использовать имя класса, как спецификатор типа данных. Например: `MAN a,b;`

```
class test {
```

```
  int a;
```

Закрытые переменные и функции

```
public:
```

```
  void set_a(int num)
```

```
  int get_a();
```

```
}
```

Открытые переменные и функции

```
void test::set_a(int num)
```

```
{ a=num; }
```

```
int test::get_a()
```

```
{ return a; }
```

```
void main()
```

```
{
```

```
  test a;
```

```
  a.set_a(10);
```

```
  printf("%d",get_a());
```

```
}
```

Два двоеточия называются оператором расширения области видимости

```

#include "stdafx.h"
#include <string>
#define N 100
using namespace std;

// Объявление стека
class Stack {
public:
    Stack() {
        stack[0] = '\0';
        stack[1] = '\0';
        stack[2] = '\0';
        stack[3] = '\0';
        stack[4] = '\0';
        stack[5] = '\0';
        stack[6] = '\0';
        stack[7] = '\0';
        stack[8] = '\0';
        stack[9] = '\0';
    }

    // Выталкивание
    char Stack::pop() {
        if (tos==0)
            cout << "Стек пуст\n";
        else
            return stack[tos--];
    }

    // Инициализация
    Stack(int i) {
        for(i=0; i<3; i++) cout << "СИМВОЛ ИЗ s1:" << s1.pop() << "\n";
        for(i=0; i<3; i++) cout << "СИМВОЛ ИЗ s2:" << s2.pop() << "\n";
        return 0;
    }

    // Помещение
    void Stack::push(char ch) {
        if (tos==SIZE-1)
            cout << "Стек переполнен\n";
        else
            stack[tos++]=ch;
    }

    // Образование двух, автоматически инициализируемых, стеков
    Stack s1, s2;
};

int main() {
    Stack s1, s2;
    int i;

    s1.push('a');
    s2.push('x');
    s1.push('b');
    s2.push('y');
    s1.push('c');
    s2.push('z');

    for(i=0; i<3; i++) cout << "СИМВОЛ ИЗ s1:" << s1.pop() << "\n";
    for(i=0; i<3; i++) cout << "СИМВОЛ ИЗ s2:" << s2.pop() << "\n";

    return 0;
}

```

```
#include "stdafx.h"
#include "iostream.h"
#include "cstring"
#include "cstdlib"
```

```
#define SIZE 255
```

```
class strtype {
```

```
    char *p;
```

```
    int len;
```

```
public:
```

```
    strtype();
```

```
    ~strtype();
```

```
    void set(char *);
```

```
    void show();
```

```
};
```

```
// Инициализация
```

```
strtype::strtype()
```

```
{
```

```
    p=(char *) new
```

```
    if(!p) {
```

```
        cout << "
```

```
        exit(1);
```

```
    }
```

```
    *p='\0';
```

```
    len=0;
```

```
}
```

```
// Освобождение памяти при удалении объекта строка
```

```
strtype::~strtype()
```

```
{
```

```
    cout << "Free p\n";
```

```
int main()
```

```
{
```

```
    strtype s1,s2;
```

```
    s1.set("This is a test");
```

```
    s2.set("I love C++");
```

```
    s1.show();
```

```
    s2.show();
```

```
    return 0;
```

```
}
```

```
    cout << p << " - length is: " << len;
```

```
    cout << "\n";
```

```
}
```

```
#include "stdafx.h"
#include "iostream.h"
#include <ctime>

class timer {
    clock_t start;
public:
    timer(); // конструктор
    ~timer(); // деструктор
};

timer::timer()
{
    start=clock();
}

timer::~~timer()
{
    clock_t end;

    end=clock();
    cout << "Time passed: " << (end-start) / CLOCKS_PER_SEC << "\n";
}

int main()
{
    timer ob;
    char c;

    // Пауза ...
    cout << "Press any key, when press ENTER: ";
    cin >> c;

    return 0;
}
```

В следующем примере приведен интересный способ использования конструктора и деструктора. В программе объект класса timer предназначен для измерения временного интервала между его созданием и удалением. При вызове деструктора на экран выводится прошедшее с момента создания объекта время.

```
#include "stdafx.h"
#include "iostream.h"
```

```
class myclass {
    int a, b;
public:
    myclass(int x, int y); // конструктор
    void show();
};
```

```
myclass::myclass(int x, int y)
{
    cout << "In constructor\n";
    a = x;
    b = y;
}
```

```
void myclass::show()
{
    cout << a << ' ' << b <<
    "\n";
}
```

```
int main()
{
    myclass ob(4, 7);

    ob.show();

    return 0;
}
```

Конструктору можно передавать параметры. Для этого добавьте необходимые параметры в объявление конструктора. Затем при объявлении объекта задайте параметры в качестве аргумента

На экране создается некоторая первичная
сущность, располагающихся на поле кл
(размер можно брать произвольн
могут как размножаться, так
при этом двум законам
Первый: если вокруг есть ровно три
особи (считая вертикали и
диагональ)
В этом случае создается новая особь. В противном
случае не создается: одни организм - этого
не достаточно, а 4,5 и более создают
популяцию

ДО КОНЦА СЕМЕСТРА!

ЖИЗНЬ

А теперь - задача.....

Запрограммировать три объекта, которые будут выполнять на игровом поле функции примитивных человечков. Два из которых будут стремиться найти друг-друга, но при этом избегать встречи с третьим. В случае если третий человечек «догонит» любого другого - тот останавливается. На игровом поле встречаются также различные препятствия, через которые ходить нельзя или ходить трудно. Такие, как камни, забор - через которые проходить нельзя и лужи, идя через которые человечек в два раза замедляет ход.

Игровое поле - массив NxM

Человек №1 - символ 1

Человек №2 - символ 2

Злой Человечек - символ @

Пустое поле - пробел

Камень - символ * (звездочка)

Забор - Символ ^ (возведение в степень)

Лужа - Символ O

Удачи!