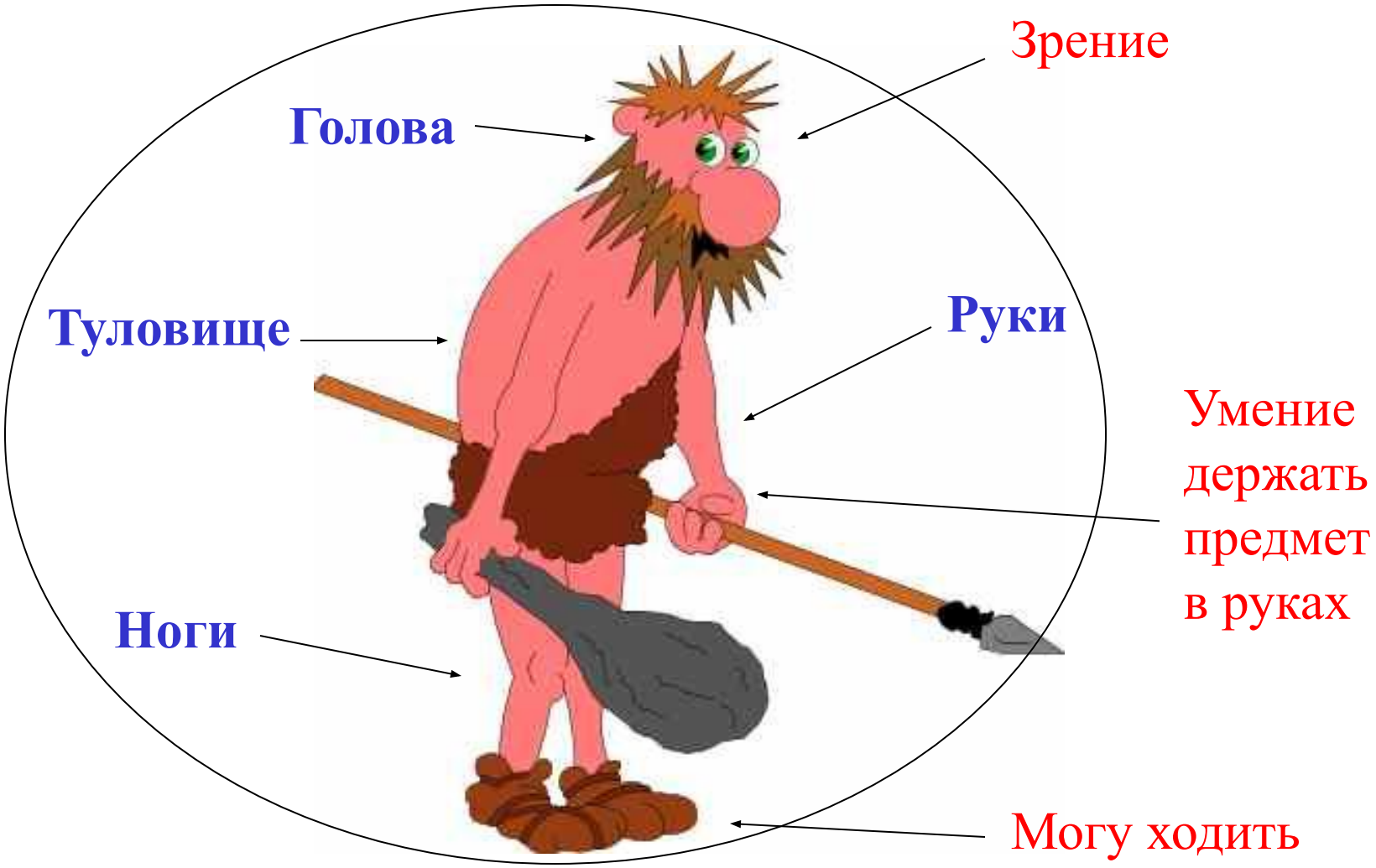
Visual C++



Microsoft Visual Studio

Человек - примитивный

Все это параметры человека

Таким образом можно сказать, что примитивный человек состоит из неких частей и способов управлять ими. Части тела могут задаваться параметрами т.е. Некоторыми переменными!

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

Добавив к параметрам их обработку получаем человека, который может выполнять примитивные функции: ходьба, обзор пространства, взятие предмета и т.д.

Параметры человека:

Позиция_x=3;
Позиция_y=4;
Движется=false;
Сила_правой_руки=10;
Сила_левой_руки=6;
Правая_рука_занята=true;
Левая_река_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!

Microsoft Visual Studio

Опишем возможные параметры человека с помощью переменных

```
Boolean Идти_вперед();
{

if (Жизненная_сила!=0)
{

        if (Позиция_свободна(x,y)=true)
                {
                  x=x+Быстрота_шага;
                  return true;
                }
        else
                {
                  Остановиться();
                  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

Рассмотрим, например, возможность идти вперед

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

Microsoft Visual Studio

Кроме инкапсуляции существует еще две основополагающих концепции объектно ориентированного программирования

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;
public:
   void set_a(int num)
   int get_a();
}

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());
}
```

Закрытые переменные и функции

Открытые переменные и функции

Два двоеточия называются оператором расширения области видимости

```
#include "stdafx.h"
#incl
#def

// Об
class
    ch
    in
publi
    sta
    vo
    ch
};

// Ин
stack
{
    co
    to
}
```

```
// Помещение
void stack::pu
{
    if (tos==SI
        cout << "
        return;
    }
    stck[tos]=c
    tos++;
}
// Выталкива
char stack::po
{
    if (tos==0)
        cout << "
        return 0;
    }  tos--;
    return stack[tos];
}
```

```
int main()
{
    // образование двух, автоматически инициализируемых,
стеков
    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;
}
```

Microsoft Visual Studio

Практически для каждого объекта требуется какого-то вида инициализация. Для этого в C++ имеется функция-конструктор. Конструктор класса вызывается каждый раз при создании объекта этого класса

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

#define SIZE 255

class strtype {
    char *p;
    int le
public:
    strtyp
    ~strt
    void
    void
};
```

```
// Инициализа
strtype::strtype
{
    p=(char *) m
    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;
}
```

```
r)

ng\n";

    cout << p << " - length is: " << len;
    cout << "\n";
}
```

Функцией обратной конструктору является деструктор.
Эта функция вызывается при удалении объекта т.к. часто
с объектом должны выполняться некоторые действия при его удалении

```
#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.

Microsoft Visual Studio

```
#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;
}
```

Microsoft Visual Studio

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
существ, располагающихся на поле кл
(размер можно брать произвольн
могут как размножаться, так
при этом двум законам
Первый: если вокр
особи (считая
диагона
в э
п

няясь

есть ровно три
вертикали и

ется новая особь. В противном
рает: одни организм - этого
но, а 4,5 и более создают
аселение

До конца семестра!

ЖИЗНЬ

А теперь - задача…..

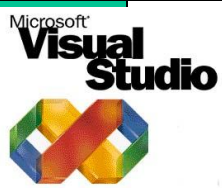
Запрограммировать три объекта, которые будут выполнять на игровом поле функции примитивных человечков.
Два из которых будут стремиться найти друг-друга, но при этом избегать встречи с третьим. В случае если третий человечек «догонит» любого другого - тот останавливается. На игровом поле встречаются также 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

Microsoft Visual Studio

Пока же можно попробовать с более простого...

Удачи!



Задание дается на две пары...