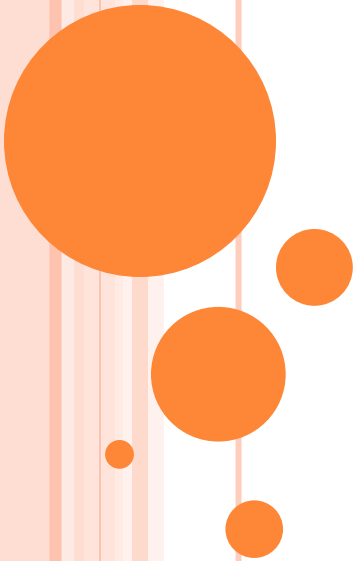


Классы

Лекция №13

Исходные предпосылки

- Необходимость объединения данных и действий над ними
- Необходимость управление доступом к переменным и функциям из различных частей программы
- Возможность мыслить в терминах предметной области
- Актуальность объектно-ориентированного подхода к программированию

Объектно-ориентированное программирование (ООП)

- класс = данные + функции (методы)
- объект = свойства + поведение

Свойства ООП:

- инкапсуляция = скрытие ненужной информации
- наследование = порождение нового объекта на основе имеющегося
- полиморфизм = перегрузка операций, функций, шаблоны классов

Класс

- Тип данных, определяемый пользователем, который объединяет описание элементов данных и функций, выполняющих действия над ними. Он включает в себя
- Элементы данных
  - имя
  - тип данных
  - режим доступа (закрытый, открытый)
- Прототипы функций
  - имя
  - типы и имена формальных параметров
  - тип результата
  - режим доступа

Прим.: Полные описания функций – в отдельном файле

Обязательные элементы класса

- Элементы данных – описания свойств объектов
- Функции:
  - конструкторы – функции для создания объектов
  - деструкторы – функции для уничтожения объектов
  - функции изменения свойств
  - функции извлечения свойств
  - функции ввода с клавиатуры
  - функции вывода на экран

Типы доступа к элементам класса

- закрытый private:
  - видимость только внутри класса
- открытый public:
  - видимость в любом месте программы

Конструкторы

- Конструктор – функция для порождения нового объекта
  - имя совпадает с именем класса
  - НЕ указывается тип результата

  - Виды конструкторов
    - По умолчанию – без входных параметров
    - С параметрами – с входными параметрами
    - Копирования – 1 входной параметр – ссылка на объект того же класса
  - В класса должен быть минимум 1 конструктор

Деструктор

- Деструктор – функция для уничтожения объекта
  - имя состоит из знака ~ и имени класса
  - НЕ указывается тип результата

Описание класса

- 2 файла
  - имя_класса.h
  - имя_класса.cpp

Формат полного описания функции класса

```
тип_результата
   имя_класса::имя(список_входных_параметров)
{

//тело функции

}

//  ::    оператор принадлежности
```

Пример описания н-файла

```
class Person
{private:
char* Fio;
int Age;
public:
Person(void)
~Person(void);
void SetFio(char* Fio1);
void SetAge(int Age1);
char* GetFio(void);
int GetAge(void);
void Input(void);
void Print(void);
};
```

Конструктор и деструктор

```
Person::Person(void)
{Fio=NULL;
Age=0;}

Person::~Person(void)
{

}
```

Функции изменения и извлечения элементов данных

```
void Person::SetFio(char *Fio1)
{int k;
k=strlen(Fio1);
Fio=new int[k];
strcpy(Fio,Fio1);
}

char* Person::GetFio(void)
{return Fio;}
```

Функции изменения и извлечения элементов данных

```
void Person::SetAge(int Age1)
{Age=Age1;}

int Person::GetAge(void)
{return(Age);}
```

Вывод на экран в ввод с клавиатуры

```
void Person::Print(void)
{printf(“%s %d”,Fio, Age);}

void Person::Input(void)
{char Buf[64];
int k;
printf(“Fio=?”);
scanf(“%s”,Buf);
k=strlen(Buf);
Fio=new char [k];
strcpy(Fio,Buf);
printf(“Age=“);
scanf(“%d”,&Age);
}
```

Пример работы с объектами класса

```
#include <stdio.h>
#include <string.h>
#include “person.h”
void main(void)
{Person A,B,C;
char *s;
int d;
A.SetFio(“Sidorov”);
A.SetAge(20);
A.Print();
B=A;
s=B.GetFio();
d=B.GEtAge();
}
```

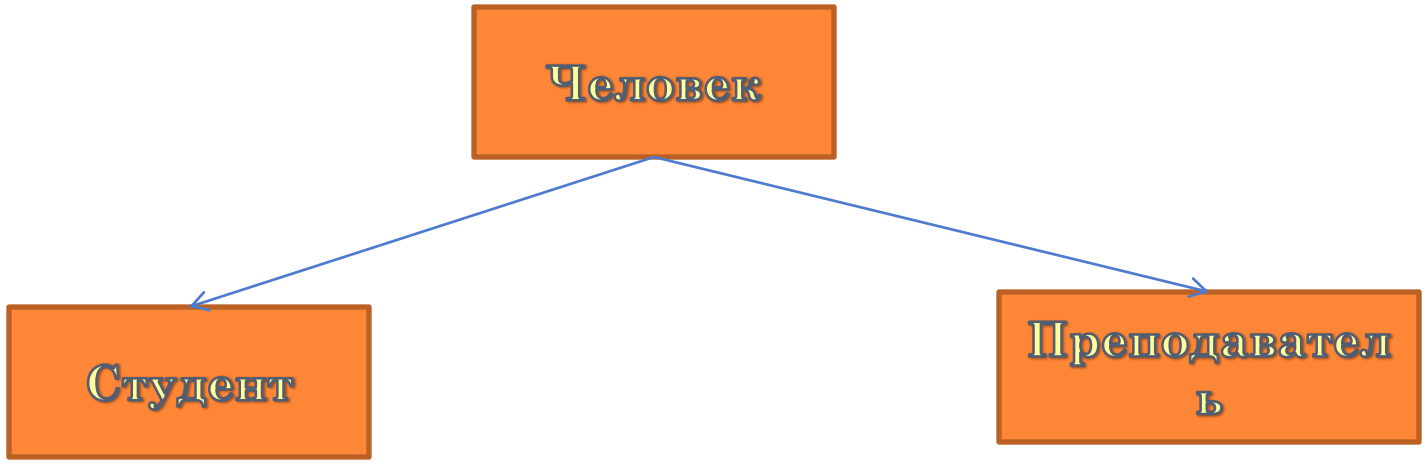
Работа с массивами объектов

- Person *pM, *q, n;
- cin>>n
- pM=new Person [n];
- for(q=pM; q<=pM+n-1; q++)
  - {q->Input();
  - q->Print ();}

- delete pM;

Наследование классов

- Образование классов на основе ранее существующих
  - от частного к общего
  - от общего к частному (построение иерархии)



Формат наследования

class имя_прозводного_класса : public имя базового класса

{

}

В базовом классе для элементов данных следует использовать доступ protected

protected:

char* Fio;

int Age;

Особенности доступа protected

- видимость внутри базового и производных классов

```
#include “person.h”
class Sotrudnik :
   public Person
{private:
   float Zarplata;
public:
   Sotrudnik(void);
   ~Sotrudnik(void);
void SetZarplata(float Zarplata1);
float GetZarplata(void);
void Input(void);
void Print(void);
};
```

- #include “sotrudnik.h”
- Sotrudnik::Sotrudnik(void)
  - {Fio=NULL; Age=0; Zarplata=0;}
- Sotrudnik::~Sotrudnik(void)
  - {}
- void Sotrudnik::SetAll (char* Fio1, int Age1, float Zarplata1)
- {Person::SetFio(Fio1);
- Age=Age1;
- Zarplata=Zarplata1;}

- void Sotrudnik::SetZaprpala(float Zarplata1)
  - {Zarplata=Zarplata1;}
- float Sotrudnik::GetZarplata(void)
  - {return Zarplata;}
- void Sotrudnik::Input(void)
- {Person::Input( );
- cin>>Zarplata;}

- void Sotrudnik::Print(void)
- {Person::Print( );
- cout<<Zarplata;}

- void main(void)
- {Sotrudnik S1;
- S1.Setname(“Sidorov”);
- S1.SetAge(25);
- S1.SetZarplata(30000.50);
- }

Переопределение операций

- 1. Можно перегружать любые операции, кроме операций ., *, ?:, ::, sizeof, универсальных для любых типов данных.
- 2. Все операции наследуются кроме присваивания.
- 3. При перегрузке операции с одним операндом она не должна иметь параметров
- 4. При перегрузке операции с двумя операндами она должна иметь один параметр (ссылку на объект).
- 5. Для организации последовательности операций они должны возвращать ссылку на объект.
- 6 Операции ++ и -- переопределяются в префиксной и постфиксной формах. Для префиксной формы входных параметров нет, для постфиксной – один целочисленный входной параметр.
- 7. Операция присваивания определяется по умолчанию.

Пример переопределения операций

```
// Сравнение объектов на меньшинство по алфавиту и по возрасту
bool Person::operator <(Person& P2)
{
    if(strcmp(Fio,P2.Fio)<0)
            return true;
    else
            if(strcmp(Fio,P2.Fio)==0 && Age<P2.Age)
                    return true;
            else
                    return false;
    return false;
}
```

```
 #include "stdafx.h"
#include ".\person.h"
 main()
{
    Person P1, P2;
    P1.SetAll("Иванов", 35);
    P2.SetAll("Петров", 20);
    if(P1<P2)
            printf("P1<P2-verno");
}
```

Переопределение операция для математических объектов

- class Vector
- {private:
  - float x,y;
- public:
  - Vector(void);
  - ~Vector(void);
  - Vector (float x1, float y1);
  - void SetAll(float x1, float y1);
  - float GetX(void);
  - float GetY(void);
  - void Print(void)
  - float GetLen(void);

- Vector operator+(Vector &V2);
- Vector operator*(float a);
- float operator*(Vector &V2);
- bool operator==(Vector &V2);
- bool operator<(Vector &V2);
- Vector* operator++(void);
- Vectror*operator++(int a);
- };

- #include <math.h>
- #include <iostream>
- #include “vector.h”
- using namespace std;
- Vector::Vector(void)
- {x=0;y=0;}

- Vector::~Vector(void)
- { }

- Vector::Vector(float x1, float y1)
- {x=x1;y=y1;}

- float Vector::GetX(void)
- {return x;}

- float Vector::GetY(void)
- {return y;}

- void Vector::Print(void)
- {cout<<“(“<<x<<“,”<<y<<“)”;}

- float GetLen(void)
- {return(sqrt(x*x+y*y))};

- Vector Vector::operator+(Vector& V2)
- {Vector V1=*this, V3;
- V3.x=V1.x+V2.x;
- ……//Дополните
- return V3;
- }

- Vector* Vector::operator++(void)
- {++x;++y;
- return this;}

- Vector* Vector::operator++(int a)
- {x++; y++;
- return this;}

- Vector Vector::operator*(float a)
- {Vector V1=*this;
- V1.x= V1.x*a;
- V1.y= V1.y*a;
- return V1;}

- float Vector::operator*(Vector &V2)
- {Vector V1=*this;
- return(V1.x*V2.x+V1.y*V2.y);}

- bool Vector::operator==(Vector &V2)
- {return(x==V2.x && y==V2.y);}

- include <vector.h>
- void main(void)
- {Vector A(1,2), B(3,4), C;
- float d,e,f;
- d=A.GetLen( );
- C=A+B;
- C.Print( );
- C=A-B; //Будет ли работать?
- C++;
- C=A*3;
- e=B*C;
- }

Выводы

- Технология ООП связывает данные с действиями над ними
- ООП обеспечивает компактность программного кода
- ООП позволяет контролировать доступ к данным