Семинар #2: Полиморфизм

Полиморфизм

Полиморфизм – это способность функций обрабатывать данные разных типов.

Статический полиморфизм

Динамический полиморфизм

Виртуальные функции

Виртуальные функции (также известные как виртуальные методы) – это механизм, который позволяет осуществлять полиморфизм. Виртуальные функции позволяют динамически определять, какая функция будет вызвана в зависимости от типа объекта, а не типа указателя или ссылки, указывающего на этот объект.

Чтобы указать, что какой-либо метод является виртуальным, нужно использовать ключевое слово `virtual` в объявлении метода в базовом классе. После этого этот метод, а также все методы с такой же сигнатурой во всех производных классах станут виртуальными. Использовать `virtual` в объявлениях методов производных классов необязательно. Виртуальный метод отличается от обычного следующим:

Если в программе возникнет ситуация когда указатель (или ссылка) на базовый класс будет указывать на объект производного класса, то при вызове виртуального метода через этот указатель (или ссылку) будет вызываться метод производного класса, а не метод базового класса.

Объект базового класса, инициализированный объектом производного

```cpp
#include <iostream>
struct Alice
{
    void say()
    {
        std::cout << "Alice" << std::endl;
    }
};

struct Bob : public Alice
{
    void say()
    {
        std::cout << "Bob" << std::endl;
    }
};

int main()
{
    Bob b;
    Alice a = b;
    a.say();  // Напечатает Alice
}
```

```cpp
#include <iostream>
struct Alice
{
    virtual void say()
    {
        std::cout << "Alice" << std::endl;
    }
};

struct Bob : public Alice
{
    void say()
    {
        std::cout << "Bob" << std::endl;
    }
};

int main()
{
    Bob b;
    Alice a = b;
    a.say();  // Всё равно напечатает Alice
}
```

Указатели на базовый класс, хранящие адрес объекта производного класса

Виртуальные функции работают в случае когда есть указатель (или ссылка) на базовый класс, который указывает на объект производного класса.

```cpp
#include <iostream>
struct Alice
{
    void say()
    {
        std::cout << "Alice" << std::endl;
    }
};

struct Bob : public Alice
{
    void say()
    {
        std::cout << "Bob" << std::endl;
    }
};

int main()
{
    Bob b;
    Alice* pa = &b;
    a->say();  // Напечатает Alice
}
```

```cpp
#include <iostream>
struct Alice
{
    virtual void say()
    {
        std::cout << "Alice" << std::endl;
    }
};

struct Bob : public Alice
{
    void say()
    {
        std::cout << "Bob" << std::endl;
    }
};

int main()
{
    Bob b;
    Alice* pa = &b;
    a->say();  // Напечатает Bob
}
```

Вызов виртуальных функций из методов класса

Ещё один случай когда работают виртуальные функции – если мы вызываем виртуальный метод из другого метода (не важно виртуального или не виртуального).

```cpp
#include <iostream>
struct Alice
{
    void say()
    {
        std::cout << "Alice" << std::endl;
    }

    void func()
    {
        say();
    }
};

struct Bob : public Alice
{
    void say()
    {
        std::cout << "Bob" << std::endl;
    }
};

int main()
{
    Bob b;
    b.func();  // Напечатает Alice
}
```

```cpp
#include <iostream>
struct Alice
{
    virtual void say()
    {
        std::cout << "Alice" << std::endl;
    }

    void func()
    {
        say();
    }
};

struct Bob : public Alice
{
    void say()
    {
        std::cout << "Bob" << std::endl;
    }
};

int main()
{
    Bob b;
    b.func();  // Напечатает Bob
}
```

Виртуальные функции в конструкторах и деструкторах

Если производный класс вызывает конструктор базового класса (это происходит автоматически в перед вызовом конструктора производного класса) и в конструкторе базового класса вызывается виртуальный метод, то вызовется метод базового класса. Это происходит просто потому что объект производного класса не может вызывать свои методы, так как он ещё не готов к этому.

```
#include <iostream>
struct Alice
{
    Alice()     {say();}
    void func() {say();}
    virtual void say()
    {
        std::cout << "Alice" << std::endl;
    }
};

struct Bob : public Alice
{
    void say()
    {
        std::cout << "Bob" << std::endl;
    }
};

int main()
{
    Bob b;     // Напечатает Alice
    b.func();  // Напечатает Bob
}
```

Похожая ситуация будет и при вызове виртуального метода в деструкторе:

```
#include <iostream>
struct Alice
{
    virtual void say() {std::cout << "Alice" << std::endl;}
    void func() {say();}
    virtual ~Alice()    {say();}
};

struct Bob : public Alice
{
    void say() {std::cout << "Bob" << std::endl;};
    ~Bob() {}
};

int main()
{
    Bob b;
    b.func();  // Напечатает Bob
               // Напечатает Alice при уничтожении объекта
}
```

Виртуальный деструктор

`override` и `final`

Полиморфизм и умные указатели

Как программа понимает, какой виртуальный метод вызывать

Размер объектов полиморфных типов. Скрытый указатель на таблицу виртуальных функций.

Абстрактные функции

Чистые виртуальные функции

Абстрактные классы и интерфейсы

Контейнер указателей на базовый класс, хранящих адрес объектов производных классов

pure virtual call и определение чистых виртуальных методов

RTTI и `dynamic_cast`

Полиморфный тип

`dynamic_cast`

`dynamic_cast` от родителя к ребёнку

`dynamic_cast` в бок

Оператор `typeid` и класс `std::type_info`

Реализация механизма виртуальных функций

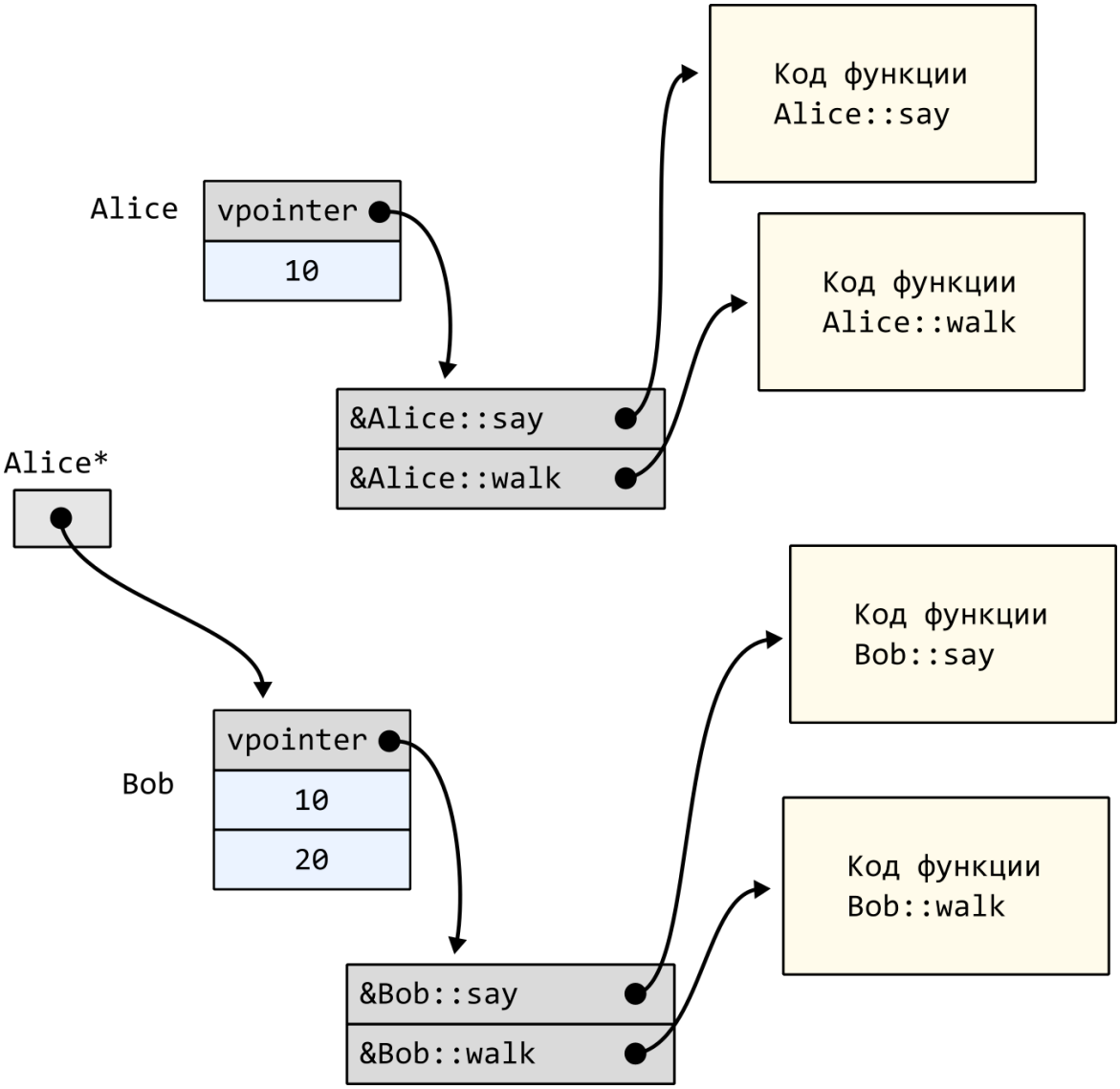
```
#include <iostream>
struct Alice
{
    virtual void say()
    {
        std::cout << "Alice Say" << std::endl;
    }
    virtual void walk()
    {
        std::cout << "Alice Walk" << std::endl;
    }
};

struct Bob : public Alice
{
    virtual void say()
    {
        std::cout << "Bob Say" << std::endl;
    }
    virtual void walk()
    {
        std::cout << "Bob Walk" << std::endl;
    }
};

int main()
{
    Alice a;
    Bob b;

    Alice* p = &a;
    p->say();  // Напечатает Alice Say

    p = &b;
    p->say();  // Напечатает Bob Say
}
```


Код функции
Alice::say
Alice
vpointer
10
Код функции
Alice::walk
&Alice::say
&Alice::walk
Alice*
Код функции
Bob::say
vpointer
Bob
10
20
Код функции
Bob::walk
&Bob::say
&Bob::walk