Раннее и позднее связывание

Как мы уже знаем из предыдущих уроков, выполнение программы в C++ происходит последовательно, строка за строкой, начиная с функции main(). Когда компилятор встречает вызов функции, то точка выполнения переходит к началу кода вызываемой функции. Как компилятор знает, что это нужно сделать?

При компиляции программы, компилятор конвертирует каждый стейтмент программы в одну или несколько строк машинного кода. Каждой строке машинного кода присваивается собственный уникальный адрес. Так же и с функциями: когда компилятор встречает функцию, она конвертируется в машинный код и получает свой адрес.

Связывание — это процесс, который используется для конвертации идентификаторов (таких как имена переменных или функций) в адреса.

Стейтмент (англ. «statement») — это наиболее распространённый тип инструкций в програм

int x — это стейтмент объявления

x = 5 — это стейтмент присваивания

std::cout << x; — это стейтмент вывода

Большинство вызовов функций, которые встречает компилятор, являются прямыми вызовами функций. Прямой вызов функции — это стейтмент, который напрямую вызывает функцию.

Прямые вызовы функций выполняются с помощью раннего связывания. Раннее связывание (или ещё «статическая привязка») означает, что компилятор (или линкер) может напрямую связать имя идентификатора (например, имя функции или переменной) с машинным адресом. Помните, что все функции имеют свой уникальный адрес. Поэтому, когда компилятор (или линкер) встречает вызов функции, он

заменяет его инструкцией машинного кода, которая сообщает процессору перейти к адресу функции.

В некоторых программах невозможно знать наперёд, какая функция будет вызываться первой. В таком случае используется позднее связывание (или ещё «динамическая привязка»). В C++ для выполнения позднего связывания используются указатели на функции. Вкратце, указатель на функцию — это тип указателя, который указывает на функцию. Функция, на которую указывает указатель, может быть вызвана через указатель и оператор вызова функции.

Вызов функции через указатель на функцию также известен как непрямой (или ещё «косвенный») вызов функции.

Позднее связывание менее эффективное, так как присутствует «посредник» между процессором и функцией. С ранним связыванием процессор может перейти непосредственно к адресу функции. С поздним связыванием процессор должен прочитать адрес, хранящийся в указателе, а затем только перейти к этому адресу. Этот дополнительный шаг и замедляет весь процесс. Однако, преимущество позднего связывания заключается в том, что оно более гибкое, нежели раннее связывание, так как не нужно решать, какую функцию следует вызывать, до, собственно, запуска самой программы.

Тем не менее, мы сталкивались с проблемой, когда родительский указатель или ссылка вызывали только родительские методы, а не дочерние.

```
#include <iostream>

class Person
{
public:
const char* getName() { return "PERSON"; }
};
```

```
class Student: public Person
{
public:
const char* getName() { return "STUDENT"; }
};

int main()
{
Student student;
Person &person = student;
std::cout << "person is a " << person.getName()<<'\n';
return 0;
}
```

Результат:

```
person is a PERSON
```

Поскольку `person` является ссылкой класса Person, то вызывается Person::getName(), хотя фактически мы ссылаемся на часть Person объекта `student`.

Виртуальные функции

Виртуальная функция в C++ — это особый тип функции, которая, при её вызове, вызывает перегруженный метод дочернего класса. Эта возможность ещё известна как **полиморфизм**.

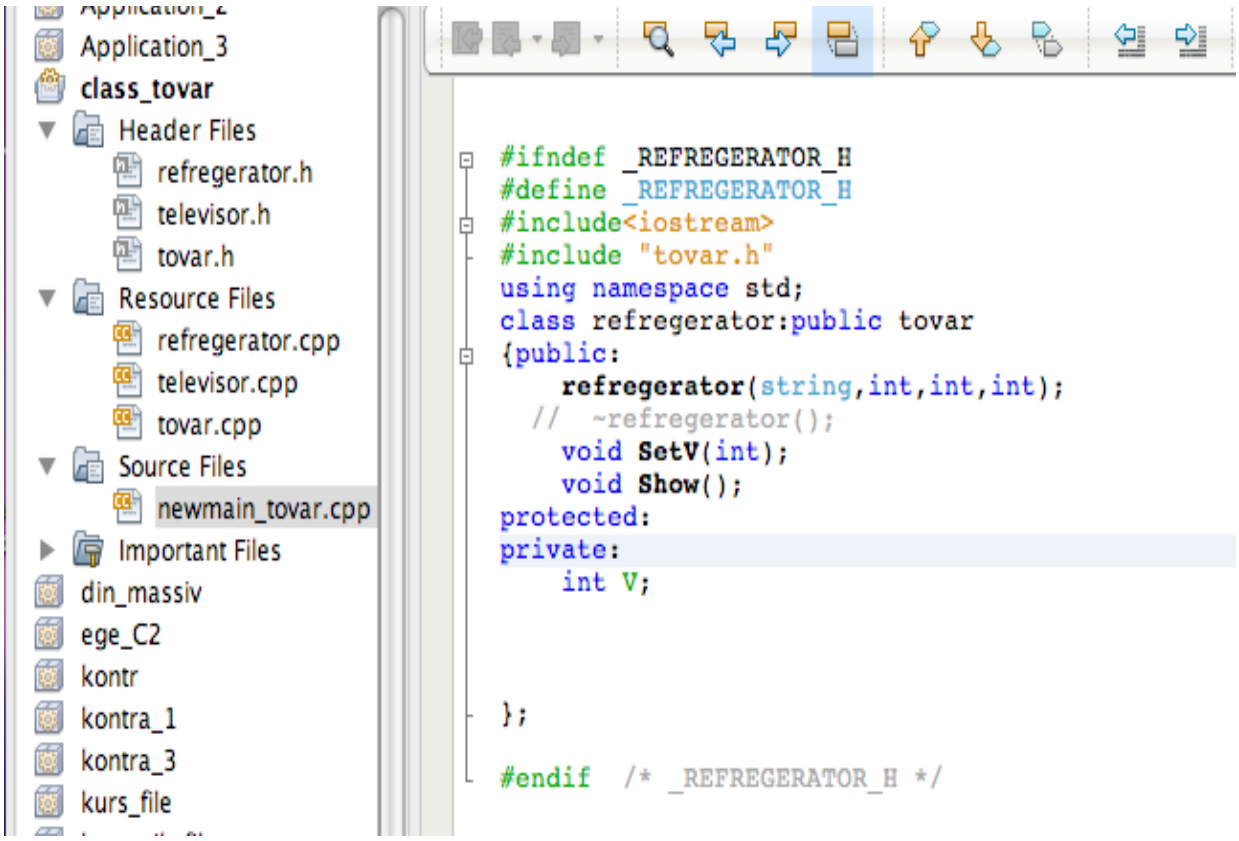
Чтобы сделать функцию виртуальной, нужно просто указать **ключевое слово virtual** перед объявлением функции. Например:

```
#include <iostream>
```

```
#include <iostream>

class Person
{
public:
virtual const char* getName() { return "PERSON"; }
};

class Student: public Person
{
public:
virtual const char* getName() { return "STUDENT"; }
};

int main()
{
Student student;
Person &person = student;
std::cout << "person is a " << person.getName()<<'\n';
return 0;
}
```

Результат:

```
person is a STUDENT
```



Пример лабораторной работы №4 (Наследование)

Задание

№ варианта	Базовый класс	Производные классы	Вычисляемый параметр
1	Товар	Телевизор, холодильник	Самый новый товар

// Файл refregerator.h

```
#ifndef _REFREGERATOR_H
#define     _REFREGERATOR_H
#include<iostream>
#include "tovar.h"
using namespace std;
class refregerator: public tovar
{
public:
```

```
    refregerator(string,int,int,int);
   //  ~refregerator();
    void SetV(int);
    void Show();
 protected:
 private:
    int V;
 };
 #endif       /* _REFREGERATOR_H */
```

// Файл televisor.h

```
#ifndef _TELEVISOR_H
#define       _TELEVISOR_H
#include <iostream>
# include "tovar.h"
using namespace std;

class televisor : public tovar
{
public:
   televisor(string, int, int, string);
  //  ~televisor();
   void SetDiagonal(string);
   void Show();
protected:
private:
   string Diagonal;

};
#endif        /* _TELEVISOR_H */
```

// Файл tovar.h

```
#ifndef _TOVAR_H
#define       _TOVAR_H
#include <iostream>
using namespace std;
```

```
class tovar {
public:
    tovar();
 //   tovar(string,int,int);
    ~tovar();
    void SetCena(int);
    void SetFirma(string);
    void SetAge(int);
    int GetAge();
    virtual void Show();
protected:
    int Cena;
    string Firma;
    int Age;

private:

};
#endif          /* _TOVAR_H */
```

// Файл refrigerator.cpp

```
#include<iostream>
#include "refregerator.h"
using namespace std;

refregerator::refregerator(string F, int C, int A, int V1)
{

    Firma=F;
    Cena=C;
    Age=A;
    V=V1;
}
//televisor::~televisor()
//{}
void refregerator::SetV(int V1)
{
 V=V1;
}

void refregerator::Show()
{
```

```
   cout<<"  Firma: "<<Firma<<" Cena: "<<Cena<<"  Age: "<<Age<<"  Obyom:
"<<V<<endl;
}
```

// Файл televisor.cpp

```
#include<iostream>
#include "televisor.h"


using namespace std;
televisor::televisor(string F, int C, int A, string D)
{

   Firma=F;
   Cena=C;
   Age=A;
   Diagonal=D;
}
//televisor::~televisor()
//{}
void televisor::SetDiagonal(string D)
{
 Diagonal=D;
}

void televisor::Show()
{
   cout<<"  Firma: "<<Firma<<" Cena: "<<Cena<<"  Age: "<<Age<<"  Diagonal:
"<<Diagonal<<endl;
}
```

// Файл tovar.cpp

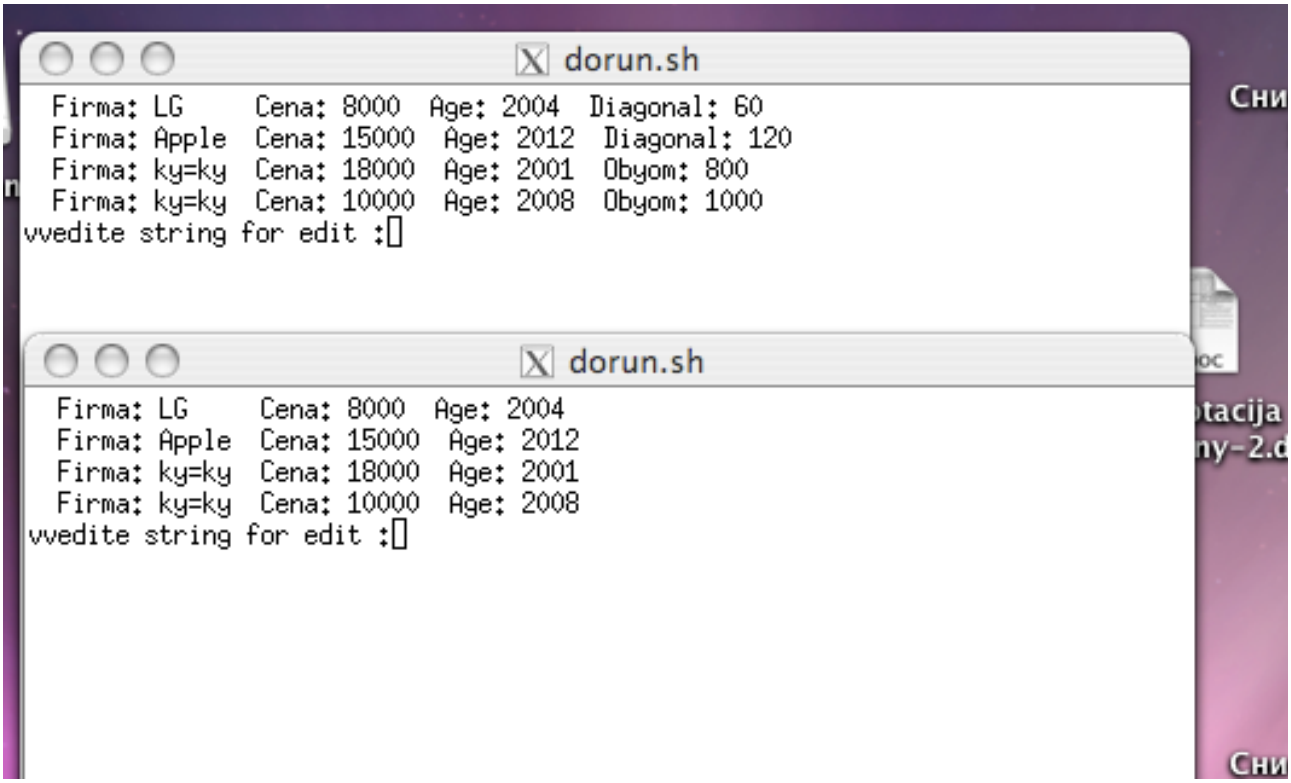
```
#include<iostream>
#include "tovar.h"
using namespace std;

tovar::tovar()
{}
//tovar::tovar(string N,string F,int A)
```

```
{
  // Name="";
  //  Firma="";
  // Age=0;
}
tovar::~tovar()
{}
void tovar::SetCena(int N)
{
   Cena=N;

}
void tovar::SetFirma(string F)
{
   Firma=F;
}
void tovar::SetAge(int A)
{
   Age=A;
}
void tovar::Show()
{
   cout<<"  Firma: "<<Firma<<" Cena: "<<Cena<<"  Age: "<<Age<<endl;
}

int tovar::GetAge()
{
   return Age;
}
```

// Файл main.cpp

```
#include <iostream>
#include "tovar.h"
#include "televisor.h"
#include "refregerator.h"
using namespace std;

int main() {
   televisor t1("LG    ",8000, 2004, "60");
   televisor t2("Apple ", 15000, 2012, "120");
   refregerator r1("ky=ky ", 18000, 2001, 800);
   refregerator r2("ky=ky ",10000, 2008, 1000);
   tovar * telref[4];
```

```
    telref[0] = &t1;
    telref[1] = &t2;
    telref[2] = &r1;
    telref[3] = &r2;
    for(int i=0;i<4;i++)
       telref[i]->Show();
    //редактирование
    int k;
    cout<<"vvedite string for edit :";
    cin>>k;
    int Age_red;

    {cin>>Age_red;
       telref[k-1]->SetAge(Age_red);

    }
    for(int i=0;i<4;i++)
       telref[i]->Show();
    int max=telref[0]->GetAge();
     for(int i=0;i<4;i++)
         if (telref[i]->GetAge()>max) max=telref[i]->GetAge();
     cout<<"max="<<max<<endl;
    return 0;

}
```


dorun.sh
Firma: LG Cena: 8000 Age: 2004 Diagonal: 60
Firma: Apple Cena: 15000 Age: 2012 Diagonal: 120
Firma: ky=ky Cena: 18000 Age: 2001 Obyom: 800
Firma: ky=ky Cena: 10000 Age: 2008 Obyom: 1000
vvedite string for edit :
dorun.sh
Firma: LG Cena: 8000 Age: 2004
Firma: Apple Cena: 15000 Age: 2012
Firma: ky=ky Cena: 18000 Age: 2001
Firma: ky=ky Cena: 10000 Age: 2008
vvedite string for edit :