Пользовательские типы данных

Лекция 10

Структуры - это

- пользовательские типы данных
- составные типы данных, построенные с использованием других типов, встроенных или пользовательских

Объявление структуры

- Объявление структуры начинается с ключевого слова `struct`. После него идет тег (`tag`) или имя структуры. Это называется заголовок структуры
- После заголовка идет тело структуры. Тело структуры заключается в фигурные скобки, после которых ставится точка с запятой
- В теле структуры прописываются поля структуры. Для каждого поля указывается тип и имя поля, уникальное в пределах структуры
- Если поля имеют один и тот же тип, их можно перечислить через запятую, указав тип один раз
- В качестве поля структуры может выступать переменная, массив или указатель

Пример объявления структуры

```
struct Time
{
   int hour;
   int minute;
   int second;
};
```

```
struct Time
{
   int hour, minute, second;
};
```

Два варианта объявления одной и той же структуры

Имя структуры

- В C++ имя структуры является именем нового типа
- В классическом Си именем нового типа будут два слова `struct + tag`
- В классическом Си имена структур принято писать заглавными буквами
- В C++ допускается имена структур начинать с большой буквы

Объявление экземпляров структуры

- После объявления структуры можно объявлять переменные типа структуры
- Переменные типа структуры называются экземпляры структуры
- Можно создать экземпляры структуры, массивы экземпляров структуры, указатели на структуру

Объявление в стиле C++

```
Time timeObject;
Time timeArray[10];
Time *timePtr;
```

Экземпляр	
Массив	
Указатель	

Объявление в стиле Си

```
struct Time timeObject;
struct Time timeArray[10];
struct Time *timePtr;
```

Объявление экземпляров структуры

Если структура объявлена со словом **typedef**, то слово **struct** в объявлении экземпляра структуры не нужно использовать (в языке Си)

```
typedef struct
{
    int hour, minute, second;
}Time;
```

```
Time t = { 3, 4, 5 };
```

Создание единственного экземпляра структуры

- Если структура не имеет тега, то можно создать единственный экземпляр такой структуры в момент ее объявления. Это аналог синглтона в языке Си.
- Если такая структура объявлена вне функций, то все ее поля по умолчанию равны 0.
- Если внутри функции, то перед использованием поля структуры необходимо проинициализировать

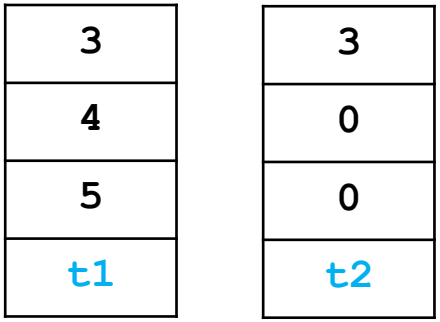
```
struct
{
    int hour, minute, second;
}Time;
```

```
Time.minute = 4;
```

Основы работы со структурой

- При объявлении структуры память не выделяется
- Память выделяется только при создании экземпляров структуры
- Все поля экземпляра структуры располагаются в памяти последовательно
- Инициализация полей структуры возможна в момент объявления экземпляра структуры

```
Time t1{ 3, 4, 5 };
Time t2{ 3 };
```



Размер экземпляра структуры

- Размер экземпляра структуры выравнивается по размеру ее самого большого поля (кратен самому большому полю)
- В C++ структура может не содержать ни одного поля
- Если структура не содержит ни одного поля, то размер ее экземпляра будет равен 1 байту

```
std::cout << sizeof(Time) << std::endl;          //12
```

```
struct Test {};
std::cout << sizeof(Test) << std::endl;          //1
```

Размер экземпляра структуры

- Размер структуры зависит от порядка следования полей структуры

```
struct Time
{
    short hour;
    int minute;
    long long second;
};
```

```
Time tt;
std::cout << sizeof(tt);
```

16

```
struct Time
{
    short hour;
    long long second;
    int minute;
};
```

```
Time tt;
std::cout << sizeof(tt);
```

24

Размер экземпляра структуры

- Для внешней структуры выравнивание определяется по размеру самого большого поля (и внутренней, и внешней структуры)

```
struct Foo {
  short iiii;
  char c;
};

struct Test1 {
  char c;
  Foo foo;
};
```

```
struct Bar {
  char c8[4];
};

struct Test2 {
  char c;
  Bar bar;
};
```

```
std::cout << sizeof(Test1);          //6
```

```
std::cout << sizeof(Test2);          //5
```

Размер экземпляра структуры

- Для внешней структуры выравнивание определяется по размеру самого большого поля (и внутренней, и внешней структуры)

```
struct Foo {
  short iiii;
  char c;
};

struct Test1 {
  double c;
  Foo foo;
};
```

```
struct Bar {
  char c8[4];
};

struct Test2 {
  char c;
  Bar bar;
};
```

```
std::cout << sizeof(Test1);          //16
```

```
std::cout << sizeof(Test2);          //5
```

Доступ к полям структуры

- Для доступа к полям структуры используются операции точка (.) и стрелка (->)
- Операция точка используется для доступа через экземпляр и через ссылку
- Операция стрелка используется для доступа через указатель

```
Time timeObject,  *timePtr;
timePtr = & timeObject;
std::cout << timePtr->hour      << ":" <<
             timePtr->minute    << ":" <<
             timePtr->second    << std::endl;
std::cout << timeObject.hour    << ":" <<
             timeObject.minute  << ":" <<
             timeObject.second  << std::endl;
```

Доступ к полям структуры

- Операции точка и стрелка взаимозаменяемы
- Для использования стрелки с экземплярами и ссылками нужно предварительно взять адрес объекта
- Для использования точки с указателями нужно предварительно разыменовать указатель

```
Time timeObject,  *timePtr;
timePtr = & timeObject;
std::cout << (*timePtr).hour       << ":" <<
             (*timePtr).minute     << ":" <<
             (*timePtr).second     << std::endl;
std::cout << (&timeObject)->hour   << ":" <<
             (&timeObject)->minute << ":" <<
             (&timeObject)->second << std::endl;
```

Вложенные структуры

Структура может быть объявлена внутри другой структуры

```
struct A
{
    struct B
    {
        int b1;
        double b2;
    };
    B b;
    double a1;
    float a2;
};
```

```
struct A
{
    struct B
    {
        int b1;
        double b2;
    }b;
    double a1;
    float a2;
};
```

Два разных варианта объявления экземпляра внутренней структуры

Вложенные структуры

- Для работы с экземплярами внутренней структуры также используются операции доступа точка и стрелка
- Инициализация экземпляра внутренней структуры проходит в рамках инициализации экземпляра внешней структуры

```
A a { 5, 3.4, 5.6, 4.5 };
std::cout << a.b.b2 << std::endl;
A *a1 = new A{ 5, 3.4, 5.6, 4.5 };
std::cout << a1->b.b2 << std::endl;
```

Вложенные структуры

- Экземпляр вложенной структуры можно создавать без создания экземпляра внешней структуры
- Для указания типа используется операция разрешения области видимости `::`

```
A::B b{ 2, 2.2 };
std::cout << b.b2 << std::endl;
A::B * b2 = new A::B{ 1, 1.1 };
std::cout << b2->b2 << std::endl;
```

Имя экземпляра структуры может совпадать с именем поля структуры

Функции и структуры

- Структуры в функцию можно передавать
  - по значению
  - по ссылке
  - по указателю
- По умолчанию структуры передаются вызовом по значению, такой способ приводит к большим накладным расходам на вызов функции
- Предпочтительнее использовать вызов по ссылке или по указателю
- Если функция не должна изменять поля передаваемой структуры, то нужно передать константную ссылку или константный указатель
- Если функция возвращает экземпляр структуры, создается его копия

Пример - структура Рациональная дробь

```
#ifndef RATIONAL_H
#define RATIONAL_H
struct  Rational
{
   int num, denum;
};
Rational add  (const Rational &, const Rational &);
Rational sub  (const Rational &, const Rational &);
Rational mult (const Rational &, const Rational &);
Rational div  (const Rational &, const Rational &);
void     print(const Rational &);
#endif
```

Заголовочный файл `rational.h`

Объявление структуры `Rational` и прототипы функций для работы с этой структурой.

Структуры передаются как константные ссылки, то есть функции не изменяют поля структур

Внутренние функции модуля Rational

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

Файл исходного кода `rational.cpp`

```
int NOD(int a, int b)
{
   if(! ( a % b ))
       return b;
   return NOD(b, a % b);
}
```

Рекурсивная функция нахождения наибольшего общего делителя (для сокращения дроби)

```
void decr(Rational & r)
{
   int a;
   a = NOD(r.num, r.denum);
   r.denum /= a;
   r.num   /= a;
}
```

Функция сокращения дроби

Внутренние функции модуля `Rational`

Файл исходного кода `rational.cpp`

Функция определения знака дроби (для печати на экране)

```
void sign(Rational & r)
{
    if(r.denum < 0 && r.num > 0)
    {
        r.denum *= -1;
        r.num   *= -1;
    }
}
```

Функция сложения рациональных дробей

Файл исходного кода
rational.cpp

```
Rational add(const Rational &left,
             const Rational &right)
{
   Rational res;
   res.num = left.num      * right.denum +
             left.denum    * right.num;
   res.denum = left.denum * right.denum;
   decr(res);
   sign(res);
   return res;
}
```

Функция создает новый экземпляр структуры и возвращает его копию, а не ссылку на него

Функция сложения двух дробей.
После сложения вызываются функции сокращения дроби и определения знака дроби.
Другие арифметические функции выглядят аналогично

Функция печати рациональной дроби

Файл исходного кода **rational.cpp**

```
void print(const Rational & r)
{
    Rational copy = r;
    decr(copy);
    sign(copy);
    if(copy.num / copy.denum)
    {
        std::cout << copy.num / copy.denum << " ";
        copy.num %= copy.denum;
    }
    if(copy.num % copy.denum==0)
    {
        std::cout << std::endl;
        return;
    }
    std::cout << copy.num << "/" << copy.denum << std::endl;
}
```

Функции decr() и sign() изменяют значение своих аргументов, поэтому нужно создать копию параметра r

Если дробь неправильная, печатаем ее целую часть и уменьшаем знаменатель

Если числитель кратен знаменателю - выходим из функции

Работа с рациональной дробью

```
#include "rational.h"
```

Файл исходного кода **test.cpp**

```
int main()
{
   Rational r1, r2, r3;
   r1.num = 45;
   r1.denum = 15;
   print(r1);
   r2.num = 2;
   r2.denum = 3;
   r3 = add(r1,r2);
   print(r3);
   r3 = sub(r1,r2);
   r3 = mult(r1,r2);
   r3 = div(r1,r2);
   r3 = r2;
   return 0;
}
```

Функция **main** содержит код, демонстрирующий работу со структурой **Rational** и функциями, объявленными в заголовочном файле **rational.h**

Одному экземпляру структуры можно присвоить другой экземпляр этой же структуры

Пример - структура «Поезд»

```
#ifndef TRAIN_H
#define TRAIN_H

#define LENGTH 100

struct Train
{
    std::string stations[LENGTH];
    std::string name;
    int countStation;
};

bool AddStation(Train & tr, std::string stationName);
bool DeleteStation(Train & tr, std::string stationName);
void PrintTrain(const Train & tr);
int FindStation(const Train & tr, std::string stationName);

#endif
```

Файл train.h

Максимально возможное количество станций у поезда

В качестве одного из полей структура содержит массив строк

Поле countStation содержит количество имеющихся остановок поезда

Прототипы функций по работе с поездом

Функции для работы с поездом

Файл train.cpp

```
#include <string>
#include "train.h"
#include <iostream>

bool AddStation(Train & tr, std::string stationName)
{
    if (tr.countStation == LENGTH)
        return false;
    tr.stations[tr.countStation++] = stationName;
    return true;
}
```

Если количество станций совпадает с максимально возможным, то больше добавить остановок не получится

Количество остановок увеличивается в момент добавления новой станции

Функции для работы с поездом

Файл train.cpp

```
bool DeleteStation(Train & tr, std::string stationName)
{
    for (int i = 0; i < tr.countStation; i++)
    {
        if (tr.stations[i] == stationName)
        {
            tr.stations[i] = tr.stations[--tr.countStation];
            return true;
        }
    }
    return false;
}
```

Если станция не найден, то функция возвращает ложь

Если название станции совпадает с искомым, то на это место записывается название последней остановки, количество станций при этом уменьшается

Функции для работы с поездом

Файл train.cpp

```
void PrintTrain(const Train & tr)
{
    std::cout << "****************************************"
              << std::endl;
    std::cout << "Train name is " << tr.name << std::endl;
    std::cout << "Count of stations is " << tr.countStation <<
    std::endl;
    std::cout << “Stations are:" << std::endl;
    for (int i = 0; i < tr.countStation; i++)
        std::cout << tr.stations[i] << std::endl;
    std::cout << "****************************************"
              << std::endl;
}
```

Функция печати не меняет полей передаваемой структуры, поэтому передается константная ссылка

Функции для работы с железнодорожной станцией

Файл train.cpp

```
int FindStation(const Train & tr, std::string stationName)
{
    for (int i = 0; i < tr.countStation; i++)
    {
        if (tr.stations[i] == stationName)
        {
            return i;
        }
    }
    return -1;
}
```

Функция возвращает индекс найденной станции или -1 если такой остановки у поезда нет

Работа со структурой

Файл test.cpp

```
#include <string>
#include "train.h"
#include <iostream>

int main()
{
    Train tr1, tr2;
    tr1.name = "first";
    tr1.countStation = 0;
    AddStation(tr1, "one");
    AddStation(tr1, "two");
    AddStation(tr1, "three");
    AddStation(tr1, "four");
    AddStation(tr1, "five");
    tr2 = tr1;
    tr2.name = "second";
    PrintTrain(tr1);
    PrintTrain(tr2);
    DeleteStation(tr1, "three");
    PrintTrain(tr1);
    PrintTrain(tr2);
    std::cout << "Index of station three
        in first train " <<
        FindStation(tr1, "three") <<
        std::endl;
    std::cout << "Index of station three
        in second train " <<
        FindStation(tr2, "three") <<
        std::endl;
    return 0;
}
```

В функции main создаются два экземпляра структуры Train, заполняется один экземпляр, затем второму присваивается значение первого. Изменяются разные поля первого и второго экземпляров, программа работает правильно

Результат работы программы

```
****************************************
Train name is first
Count of stations is 5
Stations are:
one
two
three
four
five
************************************
************************************
Train name is second
Count of stations is 5
Stations are:
one
two
three
four
five
************************************
```

Второй экземпляр инициализирован как копия первого, но изменено имя поезда

```
****************************************
Train name is first
Count of stations is 4
Stations are:
one
two
five
four
********************************
********************************
Train name is second
Count of stations is 5
Stations are:
one
two
three
four
five
****************************************
Index of station three in first train -1
Index of station three in second train 2
```

Третья станция была удалена у первого поезда, но второй поезд изменения не коснулись

Изменения в структуре «Поезд»

Теперь создадим структуру `Train` таким образом, чтобы максимальное количество станций было произвольным.

Массив станций будет динамическим, а в самой структуре для его хранения будет создан указатель

Также добавятся функции для выделения и освобождения динамической памяти при работе с экземпляром структуры

Объявление структуры и функций

Файл train.h

```
#ifndef TRAIN_H
#define TRAIN_H

struct Train
{
     std::string *stations;
     std::string name;
     int countStation;
     int maxSize;
};

void InitTrain(Train &tr);
void FreeTrain(Train &tr);
bool AddStation(Train & tr, std::string stationName);
bool DeleteStation(Train & tr, std::string stationName);
void PrintTrain(const Train & tr);
int FindStation(const Train & tr, std::string stationName);

#endif
```

Массив станций теперь хранится в виде указателя, количество выделенной под него памяти теперь является полем структуры

Функции для работы со структурой

```
#include <string>
#include "train.h"
#include <iostream>

void InitTrain(Train & tr)
{
     tr.maxSize = 10;
     tr.countStation = 0;
     tr.stations = new std::string[tr.maxSize];
}

void FreeTrain(Train & tr)
{
     if (tr.stations)
          delete[] tr.stations;
}
```

Файл train.cpp

Функция InitTrain выполняет инициализацию полей структуры начальными значениями. Также выделяет память под массив станций

Функция FreeTrain выполняет освобождение памяти из-под массива станций

Функции для работы со структурой

Файл train.cpp

```
bool AddStation(Train & tr, std::string stationName)
{
     if (tr.countStation == tr.maxSize)
     {
          std::string * temp = new std::string[tr.maxSize *= 2];
          for (int i = 0; i < tr.countStation; i++)
          {
               temp[i] = tr.stations[i];
          }
          delete[] tr.stations;
          tr.stations = temp;
     }
     tr.stations[tr.countStation++] = stationName;
     return true;
}
```

Функция добавления новой станции теперь проверяет, хватает ли места в массиве станций. Если место закончилось, то выделяется новый участок памяти в два раза большего размера, все имеющиеся станции переписываются в новый участок памяти, старая память освобождается, а указатель внутри структуры связывается с вновь выделенной памятью

Функции для работы со структурой

Файл `train.cpp`

```
bool DeleteStation(Train & tr, std::string stationName)
{
	for (int i = 0; i < tr.countStation; i++)
	{
		if (tr.stations[i] == stationName)
		{
			tr.stations[i] = tr.stations[--tr.countStation];
			return true;
		}
	}
	return false;
}

void PrintTrain(const Train & tr)
{
	std::cout << "****************************************" << std::endl;
	std::cout << "Train name is " << tr.name << std::endl;
	std::cout << "Count of stations is " << tr.countStation << std::endl;
	std::cout << "Stations are:" << std::endl;
	for (int i = 0; i < tr.countStation; i++)
		std::cout << tr.stations[i] << std::endl;
	std::cout << "****************************************" << std::endl;
}

int FindStation(const Train & tr, std::string stationName)
{
	for (int i = 0; i < tr.countStation; i++)
	{
		if (tr.stations[i] == stationName)
		{
			return i;
		}
	}
	return -1;
}
```

Все остальные функции остались без изменения

Работа со структурой

Файл test.cpp

```
#include <string>
#include "train.h"
#include <iostream>

int main()
{
    Train tr1, tr2;
    InitTrain(tr1);
    InitTrain(tr2);
    tr1.name = "first";
    tr1.countStation = 0;
    AddStation(tr1, "one");
    AddStation(tr1, "two");
    AddStation(tr1, "three");
    AddStation(tr1, "four");
    AddStation(tr1, "five");
    tr2 = tr1;
    tr2.name = "second";
    PrintTrain(tr1);
    PrintTrain(tr2);
    DeleteStation(tr1, "three");
    PrintTrain(tr1);
    PrintTrain(tr2);
    std::cout << "Index of station three
        in first train " <<
        FindStation(tr1, "three") <<
        std::endl;
    std::cout << "Index of station three
        in second train " <<
        FindStation(tr2, "three") <<
        std::endl;
    return 0;
}
```

В функции main создаются два экземпляра структуры Train, оба инициализируются, затем заполняется один экземпляр, после чего второму присваивается значение первого.
Изменяются имя второго экземпляров, а у первого удаляется третья станция

Результат работы программы

```
****************************************
Train name is first
Count of stations is 5
Stations are:
one
two
three
four
five
********************************
********************************
Train name is second
Count of stations is 5
Stations are:
one
two
three
four
five
********************************
```

Второй экземпляр инициализирован как копия первого, но изменено имя поезда

```
****************************************
Train name is first
Count of stations is 4
Stations are:
one
two
five
four
************************************
************************************
Train name is second
Count of stations is 5
Stations are:
one
two
five
four
five
****************************************
Index of station three in first train -1
Index of station three in second train -1
```

Третья станция была удалена у первого поезда, но у второго поезда третьей станции также не стало. Но количество станций у второго поезда не изменилось

Копирование сработало совсем не так, как нужно!

Работа со структурой

Файл test.cpp

```
#include <string>
#include "train.h"
#include <iostream>

int main()
{
    Train tr1, tr2;
    InitTrain(tr1);
    InitTrain(tr2);
    tr1.name = "first";
    tr1.countStation = 0;
    AddStation(tr1, "one");
    AddStation(tr1, "two");
    AddStation(tr1, "three");
    AddStation(tr1, "four");
    AddStation(tr1, "five");
    tr2 = tr1;
    tr2.name = "second";
    PrintTrain(tr1);
    PrintTrain(tr2);
    DeleteStation(tr1, "three");
    PrintTrain(tr1);
    PrintTrain(tr2);
    std::cout << "Index of station three
        in first train " <<
        FindStation(tr1, "three") <<
        std::endl;
    std::cout << "Index of station three
        in second train " <<
        FindStation(tr2, "three") <<
        std::endl;
    FreeTrain(tr1);
    FreeTrain(tr2);
    return 0;
}
```

Теперь в конце функции main появились вызовы функции FreeTrain

Результат работы программы

```
****************************************
Train name is first
Count of stations is 5
Stations are:
one
two
three
four
five
****************************************
****************************************
Train name is second
Count of statio
Stations are:
one
two
three
four
five
****************************************
```

```
****************************************
Train name is first
Count of stations is 4
Stations are:
one
two
three
four
****************************************
****************************************
Train name is second
Count of stations is 5
Stations are:
one
two
five
four
five
****************************************
Index of station three in first train -1
Index of station three in second train -1
```

```
(процесс 13556) завершает работу с кодом -1073741819.
```

Копирование сработало совсем не так, как нужно!

Почему обычное присваивание не работает

stack

stations	A1
name	“”
countStation	0
maxSize	10
tr1	

stations	B1
name	“”
countStation	0
maxSize	10
tr2	

heap

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10

После создания двух экземпляров структуры `Train` `tr1` и `tr2` и вызова для каждого из них функции `InitTrain` память программы распределена приблизительно так, как показано на рисунке

Почему обычное присваивание не работает

stack

stations	A1
name	first
countStation	5
maxSize	10
tr1	

stations	B1
name	“”
countStation	0
maxSize	10
tr2	

heap

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
one	two	three	four	five					

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10

У первого экземпляра список станций изменился

У второго экземпляра все осталось без изменений

Почему обычное присваивание не работает

stack

stations	A1
name	first
countStation	5
maxSize	10
tr1	

stations	A1
name	first
countStation	5
maxSize	10
tr2	

heap

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
one	two	three	four	five					

Поле stations экземпляра tr2 содержит тот же адрес, что и поле stations tr1 – двойная ссылка

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10

Область памяти с адресами B1-B10 оказалась потерянной для программы – утечка памяти

После присваивания каждое поле второго экземпляра tr2 получило значение поля первого экземпляра tr1

Почему обычное присваивание не работает

stack

stations	A1
name	first
countStation	5
maxSize	10
tr1	

stations	A1
name	second
countStation	5
maxSize	10
tr2	

heap

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
one	two	three	four	five					

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10

Изменение значения поля name экземпляра tr2 происходит так, как ожидалось – экземпляр tr1 эти изменения не затронули. Почему?

Почему обычное присваивание не работает

stack

stations	A1
name	first
countStation	4
maxSize	10
tr1	

stations	A1
name	second
countStation	5
maxSize	10
tr2	

heap

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
one	two	five	four	five					

Удаление третьей станции у первого поезда приводит к изменению списка станций и у `tr1`, и у `tr2`.

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10

Количество станций изменилось только у первого поезда. Почему?

Почему обычное присваивание не работает

stack

stations	A1
name	first
countStation	4
maxSize	10
tr1	

stations	A1
name	second
countStation	5
maxSize	10
tr2	

heap

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
one	two	five	four	five					

После вызова функции `FreeTrain` для первого поезда участок памяти с адресами A1-A10 считается невыделенным

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10

Почему вызов функции `FreeTrain` для второго поезда приводит к краху программы?

«Глубокое» копирование

- Присваивание структур, содержащих указатели, должно быть «глубоким».
- Оно должно включать:
  - освобождение ранее выделенной памяти,
  - выделение новой памяти
  - поэлементного заполнения новой памяти

«Глубокое» копирование

Файл train.h

```
void CopyTrain(Train & dest, const Train & source)
```

Файл train.cpp

```
void CopyTrain(Train & dest, const Train & source)
{
     if (dest.maxSize < source.maxSize)
     {
          delete[] dest.stations;
          dest.stations = new std::string[dest.maxSize =
                                          source.maxSize];
     }
     for (dest.countStation = 0;
          dest.countStation < source.countStation;
          dest.countStation++)
     {
          dest.stations[dest.countStation] =
               source.stations[dest.countStation];
     }
}
```

«Глубокое» копирование

Файл test.cpp

```
#include <string>
#include "train.h"
#include <iostream>

int main()
{
    Train tr1, tr2;
    InitTrain(tr1);
    InitTrain(tr2);
    tr1.name = "first";
    tr1.countStation = 0;
    AddStation(tr1, "one");
    AddStation(tr1, "two");
    AddStation(tr1, "three");
    AddStation(tr1, "four");
    AddStation(tr1, "five");
    CopyTrain(tr2, tr1);
    tr2.name = "second";
    PrintTrain(tr1);
    PrintTrain(tr2);
    DeleteStation(tr1, "three");
    PrintTrain(tr1);
    PrintTrain(tr2);
    std::cout << "Index of station three
        in first train " <<
        FindStation(tr1, "three") <<
        std::endl;
    std::cout << "Index of station three
        in second train " <<
        FindStation(tr2, "three") <<
        std::endl;
    FreeTrain(tr1);
    FreeTrain(tr2);
    return 0;
}
```

Присваивание заменено на функцию копирования

Результат работы программы

```
****************************************
Train name is first
Count of stations is 5
Stations are:
one
two
three
four
five
****************************
****************************
Train name is second
Count of stations is 5
Stations are:
one
two
three
four
five
****************************
```

```
****************************************
Train name is first
Count of stations is 4
Stations are:
one
two
five
four
****************************************
****************************************
Train name is second
Count of stations is 5
Stations are:
one
two
three
four
five
****************************************
Index of station three in first train -1
Index of station three in second train 2
```

Память после «глубокого» копирования

stack

stations	A1
name	first
countStation	5
maxSize	10
tr1	

stations	B1
name	second
countStation	5
maxSize	10
tr2	

heap

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
one	two	three	four	five					

В результате «глубокого» копирования каждый экземпляр по-прежнему имеет свою область памяти для хранения станций

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
one	two	three	four	five					

Память после «глубокого» копирования

stack

stations	A1
name	first
countStation	4
maxSize	10
tr1	

stations	B1
name	second
countStation	5
maxSize	10
tr2	

heap

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
one	two	five	four	five					

Удаление станции у первого экземпляра никак не отразилось на втором

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
one	two	three	four	five					

Поля битов

- Для поля структуры можно указать точное количество бит, которое оно будет занимать в памяти. Такие поля называются полями бит
- Полями бит могут быть только целочисленные типы
- Поля бит упаковываются в машинное слово
- Поля бит не могут быть организованы в массив
- От поля бит нельзя взять адрес

Поля битов

```
struct TimeAndDate
{
    unsigned hours     :5; // часы от 0 до 24
    unsigned mins      :6; // минуты от 0 до 60
    unsigned secs      :6; // секунды от 0 до 60
    unsigned weekDay   :3; // день недели
    unsigned monthDay  :6; // день месяца от 1 до 31
    unsigned month     :5; // месяц от 1 до 12
    unsigned year      :8; // год от 0 до 100
};
```

Объединения

- Объявляются так же как структуры, вместо struct используется слово union
- Все поля объединения располагаются по одному адресу
- Изменение одного поля объединения может привести к изменению значений других полей
- Размер объединения определяется самым большим полем объединения
- Используются для экономии памяти и в исследовательских целях

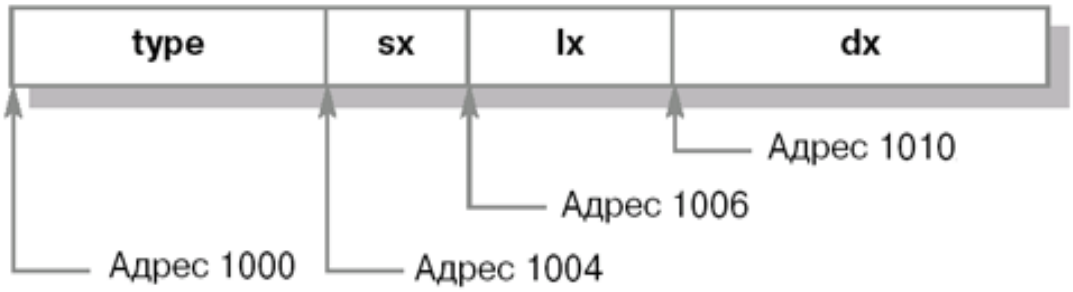
Объединения

```
struct Value {
  enum NumberType { ShortType, LongType, DoubleType };
  NumberType type;
  short sx;        // если type равен ShortType
  long lx;         // если type равен LongType
  double dx;       // если type равен DoubleType
};
```

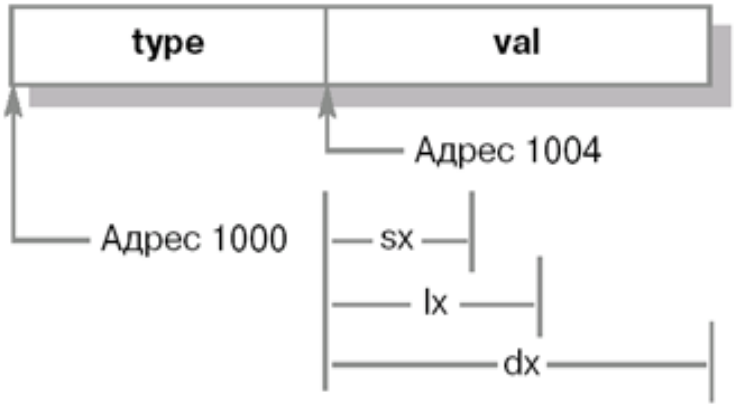
```
struct Value {
  enum NumberType { ShortType, LongType, DoubleType };
  NumberType type;
  union Number
  {
     short sx;        // если type равен ShortType
     long lx;         // если type равен LongType
     double dx;       // если type равен DoubleType
  }val;
};
```

Объединения

Расположение при первом варианте структуры



Расположение при втором варианте структуры



Объединения

```
union View
{
    float b;
    int a;
};
```

Объединение для изучения побитового представления числа с плавающей точкой

```
int main()
{
    View v;
    std::cout << "Enter a number: " << std::endl;
    std::cin >> v.b;
    char str[33];
    itoa(v.a, str, 2);
    std::cout << str << std::endl;
    std::cout << v.a << std::endl;;
    return 0;
}
```

Безымянные объединения

- Позволяют экономить память

```
int main()
{
    int d = 5;

    union
    {
        char c[8];
        double b;
        int* a;
    };

    a = &d;
    std::cout << b << std::endl;
    std::cout << a << std::endl;
    std::cout << c[2] << std::endl;
}
```

Конец