# Лабораторная работа №1

# Динамическое выделение памяти и обработка исключений в С++

***Цель работы***: 1) повторить распределение памяти в С++; 2) изучить различные способы обработки исключений; 3) получить практические навыки программирования задач с обработкой исключений.

**Порядок выполнения лабораторной работы**

1. Изучить пример из методички.
2. Запустить пример из методички и проанализировать ход работы программы.
3. Получить вариант индивидуального задания у преподавателя.
4. Реализовать полученное задание с обработкой ошибок динамического выделения памяти (**не разрешено использовать стандартную библиотеку шаблонов STL и другие готовые функции**). В программе должно быть реализовано минимум 4 исключительных ситуации.
5. Показать результат работы программы преподавателю.
6. Защитить лабораторную работу.

**Требования к отчету**

Отчет должен содержать:

1. Цель работы.

2. Задание.

3. Словесное описание исключительных ситуаций.

4. UML диаграмма классов.

5. Текст программы на языке C++ с комментариями.

6. Тесты (опии выходных форм программы).

7. Выводы.

## Теоретические сведения

**Выделение памяти**

В языке программирования C++ оператор new обеспечивает выделение динамической памяти в куче. За исключением формы, называемой «размещающей формой new», new пытается выделить достаточно памяти в куче для размещения новых данных и, в случае успеха, возвращает адрес свежевыделенной памяти. Однако, если new не может выделить память в куче, то он генерирует (throw) исключение типа std::bad\_alloc. Это устраняет необходимость явной проверки результата выделения.

Синтаксис new выглядит следующим образом:

p\_var = new typename;

где p\_var — ранее объявленный указатель типа typename. typename может подразумевать собой любой фундаментальный тип данных или объект, определенный пользователем (включая, enum, class и struct). Если typename — это тип класса или структуры, то он должен иметь доступный конструктор по умолчанию, который будет вызван для создания объекта.

Для инициализации новой переменной, созданной при помощи new нужно использовать следующий синтаксис:

p\_var = new type(initializer);

где initializer — первоначальное значение, присвоенное новой переменной, а если type — тип класса, то initializer — аргумент(ы) конструктора.

new может также создавать массив:

p\_var = new type [size];

В данном случае, size указывает размерность (длину) создаваемого одномерного массива. Адрес первого элемента возвращается и помещается вp\_var, поэтому p\_var[n] означает значение n-го элемента (считая от нулевой позиции).

Память, выделенная при помощи new, должна быть освобождена при помощи delete, дабы избежать утечки памяти. Массивы, выделенные (созданные) при помощи new[], должны освобождаться (уничтожаться) при помощи delete[].

int \*p\_scalar = new int(5);

int \*p\_array = new int[5];

Инициализаторы не могут быть указаны для массивов, созданных при помощи new. Все элементы массива инициализируются при помощи конструктора по умолчанию для данного типа. Если тип не имеет конструктора по умолчанию, выделенная область памяти не будет проинициализирована.

Существует особая форма оператора new, называемая Placement new. Данный оператор не выделяет память, а получает своим аргументом адрес на уже выделенную каким-либо образом память (например, на стеке или через malloc). Происходит размещение (инициализация) объекта путем вызова конструктора, и объект создается в памяти по указанному адресу. Часто такой метод применяют, когда у класса нет конструктора по умолчанию и при этом нужно создать массив объектов. Пример вызова выглядит следующим образом:

class A

{

public:

A(int x){}

~A(){}

};

const int n = 50;

A\* placementMemory = static\_cast<A\*>(operator new[](n \* sizeof(A)));

for (int i = 0; i < n; i++) {

new (placementMemory + i) A(rand()); //здесь память для объекта не выделяется, но инициализируется

}

//!!деинициализация памяти

for (int i = 0; i < n; i++)

{

placementMemory[i].~A();

}

operator delete[] (placementMemory);

Поскольку при выделении памяти тип создаваемого объекта(ов) не был указан, компилятор не будет вызывать деструктор для каждого объекта массива, поэтому это нужно сделать вручную, перед освобождением блока памяти.

**Проверка выделения памяти**

В компиляторах, придерживающихся стандарта ISO C++, в случае если недостаточно памяти для выделения, то генерируется исключение типа std::bad\_alloc. Выполнение всего последующего кода прекращается, пока ошибка не будет обработана в блоке try-catch или произойдет экстренное завершение программы. Программа не нуждается в проверке значения указателя; если не было сгенерировано исключение, то выделение прошло успешно. Реализованные операции определяются в заголовке <new>. В большинстве реализаций C++ оператор new также может быть перегружен для определения особого поведения.

**Освобождение памяти**

В языке программирования C++ оператор delete (или delete[]) возвращает память, выделенную оператором new, обратно в кучу. Вызов delete должен происходить для каждого вызова new, чтобы избежать утечки памяти. После вызова delete объект, указывающий на этот участок памяти, становится некорректным и не должен больше использоваться. Многие программисты присваивают 0 (нуль-указатель) указателям после использования delete, чтобы минимизировать количество ошибок программирования. Однако нужно отметить, что удаление нуль-указателя фактически не имеет эффекта, так что нет необходимости проверять нуль-указатель перед вызовом delete.

Фрагмент кода в качестве примера:

int \*p\_var = NULL; // объявление нового указателя

p\_var = new int; // память динамически выделяется

/\* .......

остальной код

........\*/

delete p\_var; // память освобождается

p\_var = NULL; // указатель заменяется на0 (нуль-указатель)

Массивы, созданные(выделенные) при помощиnew [], аналогичным образом должны быть уничтожены

(оcвобождены) при помощиdelete []:

int size = 10;

int \*p\_var = NULL; // объявление нового указателя

p\_var = new int [size];// память динамически выделяется

/\* .......

остальной код

........\*/

delete [] p\_var; // память освобождается

p\_var = NULL; // указатель заменяется на0 (нуль-указатель)

Вызов delete[] для массива объектов приведет к вызову деструктора для каждого объекта перед освобождением памяти, выделенной под массив.

Исключения - возникновение непредвиденных ошибочных ситуаций, например, деление на ноль при операциях с плавающей точкой. Обычно эти условия завершают программу пользователя с системным сообщением об ошибке. Обработка исключений в С++ дает возможность программисту восстанавливать программу из этих условий и продолжать ее выполнение.

### Исключения в C++

Язык С++ имеет чувствительный к контексту механизм обработки особых ситуаций.

Контекст для установки исключения - это блок **try**.

Обработчики объявлены в конце блока try с использованием ключевого слова **catch**.

Простой пример:

vect::vect(int n)

{ if (n < 1)

throw(n);

p = new int[n];

if (p == 0)

throw("FREE STORE EXHAUSTED");

}

void g()

{ try { vect a(n), b(n);

...

}

catch(int n) { ... } //отслеживает все неправильные размеры

catch(char\* error) {...} //отслеживает превышение свободной памяти

}

### Установленные исключения

Синтаксически выражения throw возникает в двух формах:

throw;

throw выражение;

Выражение throw устанавливает исключение. Выражение throw без аргумента повторно устанавливает текущее исключение. Обычно оно используется, когда для дальнейшей обработки исключения необходим второй обработчик, вызываемый из первого.

void foo()

{ int i;

...

throw (i);

}

main()

{ try {

foo();

}

catch(int i) { ... }

}

Если пользователь хочет выводить дополнительную информацию или использовать ее для принятия решения относительно действий обработчика, то допустимо формирование в виде объекта.

enum error {bounds, heap, other};

class vect\_error

{ private:

error e\_type;

int ub, index, size;

public:

vect\_error(error, int, int); //пакет вне заданных пределов

vect\_error(error, int); //пакет вне памяти

}

Теперь выражение throw может быть более информативным

...

throw vect\_error(bounds, i, ub);

...

### Блоки try

Синтаксически блок try имеет такую форму

*try*

*составной оператор*

*список обработчиков*

Блок try - контекст для принятия решения о том, какие обработчики вызываются для установленного исключения.

try {

...

throw("SOS");

...

io\_condition.eof(argv[i]);

throw(eof);

...

}

catch (const char\*) {...}

catch (io\_condition& x) {...}

Выражение throw соответствует аргументу catch, если он:

* точно соответствует.
* общий базовый класс порожденного типа представляет собой то, что устанавливается.
* объект установленного типа является типом указателя, преобразуемым в тип указателя, являющегося аргументом catch.

### Обработчики catch

Синтаксически обработчик catch имеет следующую форму

*catch (формальный аргумент)*

*составной оператор*

catch (char\* message)

{ cerr << message << endl;

}

catch (...) //действие по умолчанию

{

cerr << "THAT'S ALL FOLKS." << endl;

abort();

}

### Спецификация исключения

Синтаксис

*Заголовок\_функции throw(список типов)*

void foo() throw(int, over\_flow);

void noex(int i) throw();

### Terminate() и unexpected()

Обработчик terminate() вызывается, когда для обработки исключения не поставлен другой обработчик. По умолчанию вызывается функция abort().

Обработчик unexpected() вызывается, когда исключения не было в списке спецификации исключения

### Пример кода, реализующего исключение

*Главный модуль*

#include "vect.h"

using namespace std;

void g(int n)

{ try {

//блок try - контекст для принятия решения о том, какие

//обработчики вызываются для установленного исключения

vect a(n);

}

catch (int n) // обработчик

{

cerr<< "SIZE ERROR " << n <<endl;

g(10);

}

catch (const char\* error) // обработчик

{

cerr<< error <<endl;

abort();

}

}

void main()

{

extern void g(int n);

g(-1);

system("pause");

}

*Файл vect.h:*

#include <iostream>

class vect

{ private:

int\* p;

int size;

public:

vect() { size = 11; p = new int[size]; }

vect(int n);

~vect() { delete [] p; }

int& element(int i);

int ub() const { return (size - 1); }

};

vect::vect(int n)

{ if(n< 1) // оговоренное предусловие

throw (n); // устанавливается исключение

p = new int[n];

if(p == 0) // оговоренное постусловие

throw ("FREE STORE EXHAUSTED"); // устанавливается

// исключение

}

int&vect::element(int n)

{ if(n < 0 || n > size-1)

throw ("ILLEGAL NUMBER OF ELEMENT");

// устанавливается исключение

return (p[n]);

}

**Контрольные вопросы**

1. Какую цель преследует использование в программе обработки исключений?
2. Как оформляется блок обработки исключений?
3. Что такое обработчики исключений?

## Варианты заданий

| **Номер варианта** | **Задание** |
| --- | --- |
| **1** | Реализуйте класс для хранения целых чисел без знака. Опишите метод умножения двух целых, обработайте ошибку переполнения сверху (overflow). |
| **2** | Реализуйте класс для хранения целых чисел без знака . Опишите метод деления двух целых, обработайте ошибку переполнения снизу (underflow). |
| **3** | Реализуйте класс для хранения целых чисел без знака. Реализуйте метод сложения двух целых. Породите и обработайте ошибку переполнения сверху (overflow). |
| **4** | Реализуйте класс для хранения целых чисел без знака . Опишите метод деления двух целых, обработайте ошибку деления на ноль (zero division). |
| **5** | Реализуйте класс «массив целых чисел». Переопределите оператор ++ для указателя на массив целых, обработайте ошибку выхода за границы массива. |
| **6** | Реализуйте класс для хранения номера телефона. Опишите метод анализа номера телефона, обработайте ошибки динамического выделения памяти, обработайте ошибку задания номера в неверном формате (допустимый формат - +7(095)555-44-33). |
| **7** | Реализуйте класс «список целых чисел». Обработайте ошибки динамического выделения памяти. Перегрузите оператор [] и обработайте ошибку выхода за границы списка. |
| **8** | Реализуйте класс для хранения даты. Опишите метод, возвращающий день недели. Обработайте ошибки динамического выделения памяти. Обработайте ошибки неверного дня или месяца. |
| **9** | Реализуйте класс «числа с плавающей запятой». Опишите метод умножения двух чисел с плавающей запятой, обработайте ошибку переполнения сверху (overflow). |
| **10** | Реализуйте класс «числа с плавающей запятой». Опишите метод деления двух чисел с плавающей запятой, обработайте ошибку переполнения снизу (underflow). |
| **11** | Реализуйте класс «числа с плавающей запятой». Опишите метод деления двух чисел с плавающей запятой, обработайте ошибку деления на ноль (zero division). |
| **12** | Реализуйте класс «очередь» из строк. Реализуйте методы для вставки в очередь и удаления. Породите и обработайте ошибки динамического выделения памяти, переполнения очереди. |
| **13** | Реализуйте класс «очередь» из целых чисел. Опишите оператор [] для очереди элементов, обработайте ошибку выхода за границы очереди. |
| **14** | Реализуйте класс «вектор» из целых чисел. Опишите оператор [] для вектора элементов, обработайте ошибку выхода за границы массива. |
| **15** | Реализуйте класс «двоичное дерево». В качестве ключа для распределения данных используйте целые числа, в качестве самих данных– вещественные числа. Обработайте ошибки динамического выделения памяти. Переопределите оператор [] для доступа к данным дерева по значению ключа, породите и обработайте ошибку отсутствия нужных элементов в дереве. |
| **16** | Реализуйте класс для хранения личных данных (ФИО). Обработайте ошибки динамического выделения памяти. Обработайте ошибки задания ФИО (допустимый формат – «только буквы, без некоторых букв вначале слова»). |
| **17** | Реализуйте класс для хранения сетевого адреса в формате IPv4. Обработайте ошибки динамического выделения памяти. Обработайте ошибки задания адреса (допустимый формат– четыре числа от 0 до 255, разделенные точками, исключая некоторые недопустимые комбинации). |
| **18** | Реализуйте класс для хранения цепных дробей (список неповторяющихся целых чисел). Опишите методы для вставки нового числа и нахождения i-го числа. Породите и обработайте ошибки динамического выделения памяти и вставки неверного числа. |
| **19** | Реализуйте класс «битовая карта» на основе целых чисел. Опишите оператор [] для определения i-го бита. Породите и обработайте ошибку выхода за границы карты. |
| **20** | Реализуйте класс для хранения массива географических координат. Породите и обработайте ошибки динамического выделения памяти и ошибки задания неверной широты и/или долготы. |
| **21** | Реализовать класс для хранения двоичных чисел. Реализовать метод перевода числа из двоичной системы счисления в троичную. Проверить правильность перевода числа из двоичной системы счисления в троичную. |
| **22** | Реализовать класс для хранения троичных чисел. Реализовать метод перевода числа из троичной системы счисления в четырнадцатеричную. Проверить правильность перевода числа из троичной системы счисления в четырнадцатеричную. |
| **23** | Реализовать класс для хранения четверичных чисел. Реализовать метод перевода числа из четверичной системы счисления в пятеричную. Проверить правильность перевода числа из четверичной системы счисления в пятеричную. |
| **24** | Реализовать класс для хранения восьмеричных чисел. Реализовать метод перевода числа из восьмеричной системы счисления в шестеричную. Проверить правильность перевода числа из восьмеричной системы счисления в шестеричную. |
| **25** | Реализовать класс для хранения двоичных чисел. Реализовать метод правильность перевода числа из двоичной системы счисления в семеричную. Проверить правильность перевода числа из двоичной системы счисления в семеричную. |
| **26** | Реализовать класс для хранения тринадцатеричных чисел. Реализовать метод перевода числа из тринадцатеричной системы счисления в восьмеричную. Проверить правильность перевода числа из тринадцатеричной системы счисления в восьмеричную. |
| **27** | Реализовать класс для хранения одиннадцатеричных чисел. Реализовать метод перевода числа из одиннадцатеричной системы счисления в девятиричную. Проверить правильность перевода числа из одиннадцатеричной системы счисления в девятиричную. |
| **28** | Реализовать класс для хранения семеричных чисел. Реализовать метод перевода числа из семеричной системы счисления в одиннадцатеричную. Проверить правильность перевода числа из семеричной системы счисления в одиннадцатеричную. |
| **29** | Реализовать класс для хранения троичных чисел. Реализовать метод перевода числа из восемнадцатеричной системы счисления в девятнадцатеричную. Проверить правильность перевода числа из восемнадцатеричной системы счисления в девятнадцатеричную. |
| **30** | Реализовать класс для хранения двенадцатеричных чисел. Реализовать метод перевода числа из двенадцатеричной системы счисления в тринадцатеричную. Проверить правильность перевода числа из двенадцатеричной системы счисления в тринадцатеричную. |