



ОБЪЕКТНО-  
ОРИЕНТИРОВАННОЕ  
ПРОГРАММИРОВАНИЕ

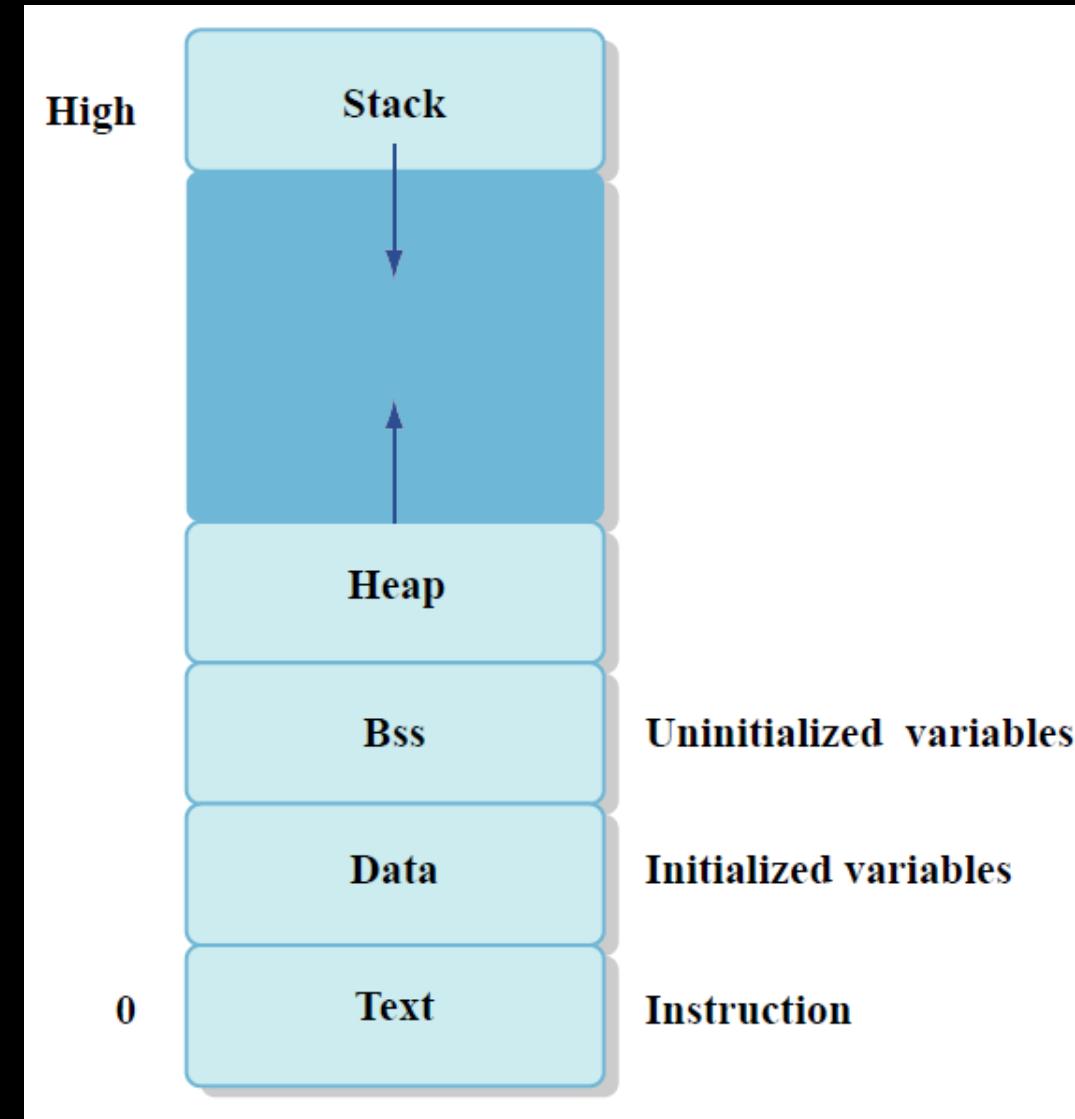
2024

# ПЛАН ЗАНЯТИЯ

- Изучаем работы с памятью
- Указатели
- Массивы
- Ссылки
- Исключения
- Классы

# ВЫДЕЛЕНИЕ ПАМЯТИ В СТЕКЕ (STACK)

- ФУНКЦИИ С РАЗМЕЩАЮТСЯ В СТЕКЕ:
  - ФУНКЦИИ ПОМЕЩАЮТСЯ В СТЕК, В МОМЕНТ ВЫЗОВА.
  - ФУНКЦИИ УДАЛЯЮТСЯ ИЗ СТЕКА В МОМЕНТ КОГДА ВЫЗЫВАЕТСЯ RETURN.
  - ФУНКЦИЯ МОЖЕТ ИСПОЛЬЗОВАТЬ ЛЮБУЮ ПАМЯТЬ В ПРЕДЕЛАХ СТЕКА.



# РУЧНОЕ УПРАВЛЕНИЕ ПАМЯТЬЮ В C++

- ПОЗВОЛИТЬ ПРОГРАММЕ ПОМЕЧАТЬ ОБЛАСТИ ПАМЯТИ КАК «ЗАНЯТЫЕ» ПОЛЕЗНОЙ ИНФОРМАЦИЕЙ.
- ПОЗВОЛИТЬ ПРОГРАММЫ ПОМЕЧАТЬ ОБЛАСТИ ПАМЯТИ КАК «НЕ ЗАНЯТЫЕ» ПО ОКОНЧАНИИ РАБОТЫ.  
ЧТО БЫ ЭТИ ОБЛАСТИ МОГЛИ ИСПОЛЬЗОВАТЬ ДРУГИЕ АЛГОРИТМЫ И ПРОГРАММЫ.

## УПРАВЛЕНИЕ ПАМЯТЬЮ: КУЧА (HEAP)

- КУЧА – ЭТО ОБЛАСТЬ ПАМЯТИ, КОТОРЫЙ МОЖЕТ ИСПОЛЬЗОВАТЬ ПРОГРАММА.
- КУЧУ МОЖНО СРАВНИТЬ С ГИГАНТСКИМ МАССИВОМ.
- ДЛЯ РАБОТЫ С КУЧЕЙ ИСПОЛЬЗУЕТСЯ СПЕЦИАЛЬНЫЙ СИНТАКСИС УКАЗАТЕЛЕЙ.
- ВСЯ ПРОГРАММА МОЖЕТ ПОЛУЧИТЬ ДОСТУП К КУЧЕ.

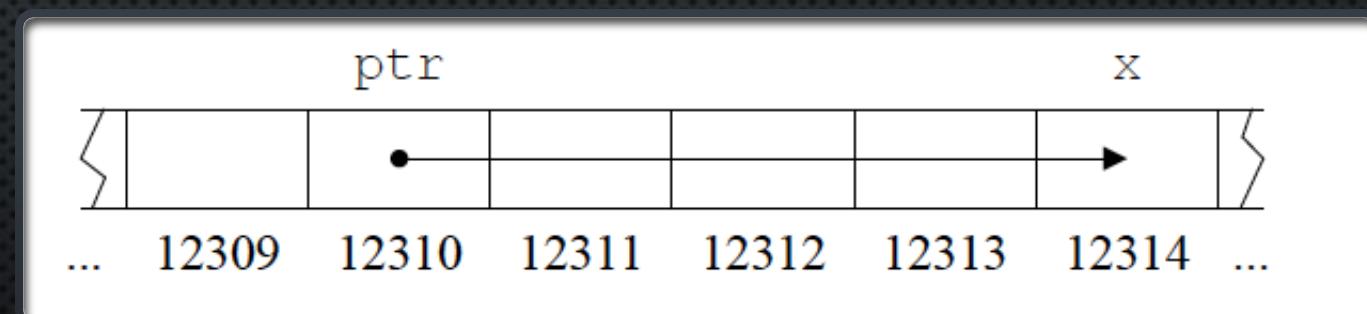
Addr.	Contents
:	:
0xbeef	0xbeef
0xbf4	0xfeed
:	:

# УКАЗАТЕЛЬ ХРАНИТ АДРЕС ПЕРЕМЕННОЙ!

**INT X = 5;**

**INT \* PTR = &X;**

# УКАЗАТЕЛЬ – ЭТО ЧИСЛО



# ПУСТОЙ УКАЗАТЕЛЬ **NULLPTR**

- РАНЬШЕ, ДЛЯ ОБНУЛЕНИЯ УКАЗАТЕЛЕЙ ИСПОЛЬЗОВАЛСЯ МАКРОС **NUL**, ЯВЛЯЮЩИЙСЯ НУЛЕМ — ЦЕЛЫМ ТИПОМ, ЧТО, ЕСТЕСТВЕННО, ВЫЗЫВАЛО ПРОБЛЕМЫ (НАПРИМЕР, ПРИ ПЕРЕГРУЗКЕ ФУНКЦИЙ).
- КЛЮЧЕВОЕ СЛОВО **NULLPTR** ИМЕЕТ СВОЙ СОБСТВЕННЫЙ ТИП **STD::NULLPTR\_T**, ЧТО ИЗБАВЛЯЕТ НАС ОТ БЫВШИХ ПРОБЛЕМ.
- СУЩЕСТВУЮТ НЕЯВНЫЕ ПРЕОБРАЗОВАНИЯ NULLPTR К НУЛЕВОМУ УКАЗАТЕЛЮ ЛЮБОГО ТИПА И К BOOL.



## ПРИМЕРЫ

- 01\_DECLARINGANDUSINGPOINTERS
- 02\_POINTERTOCHAR
- 03\_CONSTPOINTERANDPOINTERTOCONST
- 04\_POINTERSANDARRAYS
- 05\_POINTERARITHMETIC\_NAVIGATION
- 06\_POINTERARITHMETIC\_DISTANCEBETWEENELEMENTS
- 07\_DYNAMICMEMORYALLOCATION
- 08\_DANGLINGPOINTERS
- 09\_WHENNEWFAILS
- 10\_MEMORYLEAKS
- 11\_DYNAMICALLYALLOCATEDARRAYS

# ПЕРЕДАЧА ИНФОРМАЦИИ О ПЕРЕМЕННЫХ В ФУНКЦИИ И ОБЪЕКТЫ

Из С мы помним, что передавать переменные можно:

- «по значению» – путем копирования;
- «с помощью указателя» – тогда копируется указатель, а переменная «остается на месте»;

# &ССЫЛКИ

// СТРУКТУРА ОБЪЯВЛЕНИЯ ССЫЛОК

/\*тип\*/ &/\*имя ссылки\*/ = /\*имя переменной\*/;

## & ССЫЛКИ

- ✓ Ссылка — это **синоним** имени переменной, т. е. другое имя для использования переменной.
- ✓ Отличие ссылки от указательной переменной в том, что ссылка **не является объектом**. Для названия ссылки может не отводиться место в памяти. Для указательной переменной место в памяти выделяется всегда.
- ✓ Ссылка **не может ссылаться на несуществующий объект**, указатель может.
- ✓ Ссылку нельзя переназначить, а указательную переменную можно.

# LVALUE & RVALUE ПЕРЕМЕННЫЕ

- С КАЖДОЙ **ОБЫЧНОЙ** ПЕРЕМЕННОЙ СВЯЗАНЫ ДВЕ ВЕЩИ – **АДРЕС** И **ЗНАЧЕНИЕ**.
- `INT I; // СОЗДАТЬ ПЕРЕМЕННУЮ ПО АДРЕСУ, НАПРИМЕР 0x10000`
- `I = 17; // ИЗМЕНИТЬ ЗНАЧЕНИЕ ПО АДРЕСУ 0x10000 НА 17`
- А ЧТО БУДЕТ ЕСЛИ У МЕНЯ ЕСТЬ ТОЛЬКО ЗНАЧЕНИЕ? Могу ли я СДЕЛАТЬ ТАК: `20=10;` ?

# С ЛЮБЫМ ВЫРАЖЕНИЕМ СВЯЗАНЫ ЛИБО АДРЕС И ЗНАЧЕНИЕ, ЛИБО ТОЛЬКО ЗНАЧЕНИЕ

- Для того, чтобы отличать выражения, обозначающие объекты, от выражений, обозначающих только значение, ввели понятия **LVALUE** и **RVALUE**.
- Изначально слово **LVALUE** использовалось для обозначения выражений, которые могли стоять слева от знака присваивания (*LEFT-VALUE*); им противопоставлялись выражения, которые могли находиться только справа от знака присваивания (*RIGHT-VALUE*).

---

i — lvalue

---

+i — lvalue

---

\*&i — lvalue

---

a[5] — lvalue

---

a[i] — lvalue

---

10 — rvalue

---

i + 1 — rvalue

---

i++ — rvalue

## ПРИМЕР



## ПРИМЕРЫ

- 12\_DECLARINGANDUSINGREFERENCES
- 13\_COMPAREPOINTERSANDREFERENCES
- 14\_REFERENCESANDCONST
- 15\_REFERENCESWITHRANGEBASEDFORLOOPS
- 16\_LVALUEANDRVALUE

# КАК ОБРАБАТЫВАТЬ АЛГОРИТМИЧЕСКИЕ ОШИБКИ?

1. ВЕРНУТЬ РЕЗУЛЬТАТ ОПЕРАЦИИ ЯВНО (ОПЕРАЦИЯ УСПЕШНА / ОПЕРАЦИЯ НЕ УСПЕШНА)
2. ВЕРНУТЬ РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ОПЕРАЦИИ КАК ОДИН ИЗ ПАРАМЕТРОВ (ПО ССЫЛКЕ)

# ПРИМЕР

17\_RETURNERROR

# EXCEPTIONS

ДЛЯ РЕАЛИЗАЦИИ МЕХАНИЗМА ОБРАБОТКИ ИСКЛЮЧЕНИЙ В ЯЗЫК Си++ ВВЕДЕНЫ СЛЕДУЮЩИЕ ТРИ КЛЮЧЕВЫХ (СЛУЖЕБНЫХ) СЛОВА:

1. **TRY** (КОНТРОЛИРОВАТЬ)
2. **CATCH** (ЛОВИТЬ)
3. **THROW** (ГЕНЕРИРОВАТЬ, ПОРОЖДАТЬ, БРОСАТЬ, ПОСЫЛАТЬ, ФОРМИРОВАТЬ).

# EXCEPTION

ИСКЛЮЧЕНИЕ ЭТО:

1. ОБЪЕКТ, НАСЛЕДНИК КЛАССА STD::EXCEPTION (#INCLUDE <EXCEPTION>)
2. СОБЫТИЕ, ПРЕРЫВАЮЩЕЕ ОБРАБОТКУ ПРОГРАММЫ.

ПОД ПРЕРЫВАНИЕМ МЫ ПОНИМАЕМ, ЧТО СРАБАТЫВАНИЕ ИСКЛЮЧЕНИЯ, АНАЛОГИЧНО СРАБАТЫВАНИЮ ОПЕРАТОРА **RETURN**.

НО ЕСТЬ СУЩЕСТВЕННОЕ ОТЛИЧИЕ: **RETURN** ВОЗВРАЩАЕТ РЕЗУЛЬТАТ В ТО МЕСТО, ГДЕ ВЫЗВАЛИ ФУНКЦИЮ.

ИСКЛЮЧЕНИЕ ВОЗВРАЩАЕТ ОБЪЕКТ ИСКЛЮЧЕНИЯ ТОЛЬКО В ТЕ МЕСТА, ГДЕ ЕГО ЯВНО ЛОВЯТ (**CATCH**)!

И ТОЛЬКО ЕСЛИ ИСКЛЮЧЕНИЕ СРАБОТАЛО (**THROW**) В МЕСТЕ ГДЕ МЫ ЕГО КОНТРОЛИРУЕМ (**TRY**)!

ЕСЛИ ИСКЛЮЧЕНИЕ НЕ ПОЙМАТЬ (**CATCH**) ТО ОНО БУДЕТ ПРЕРЫВАТЬ РАБОТУ ФУНКЦИЙ, ПОДНИМАЯСЬ ВВЕРХ ПО СТЕКУ ВЫЗОВА, ПОКА НЕ ОСТАНОВИТ ПРОГРАММУ.

# TRY / CATCH

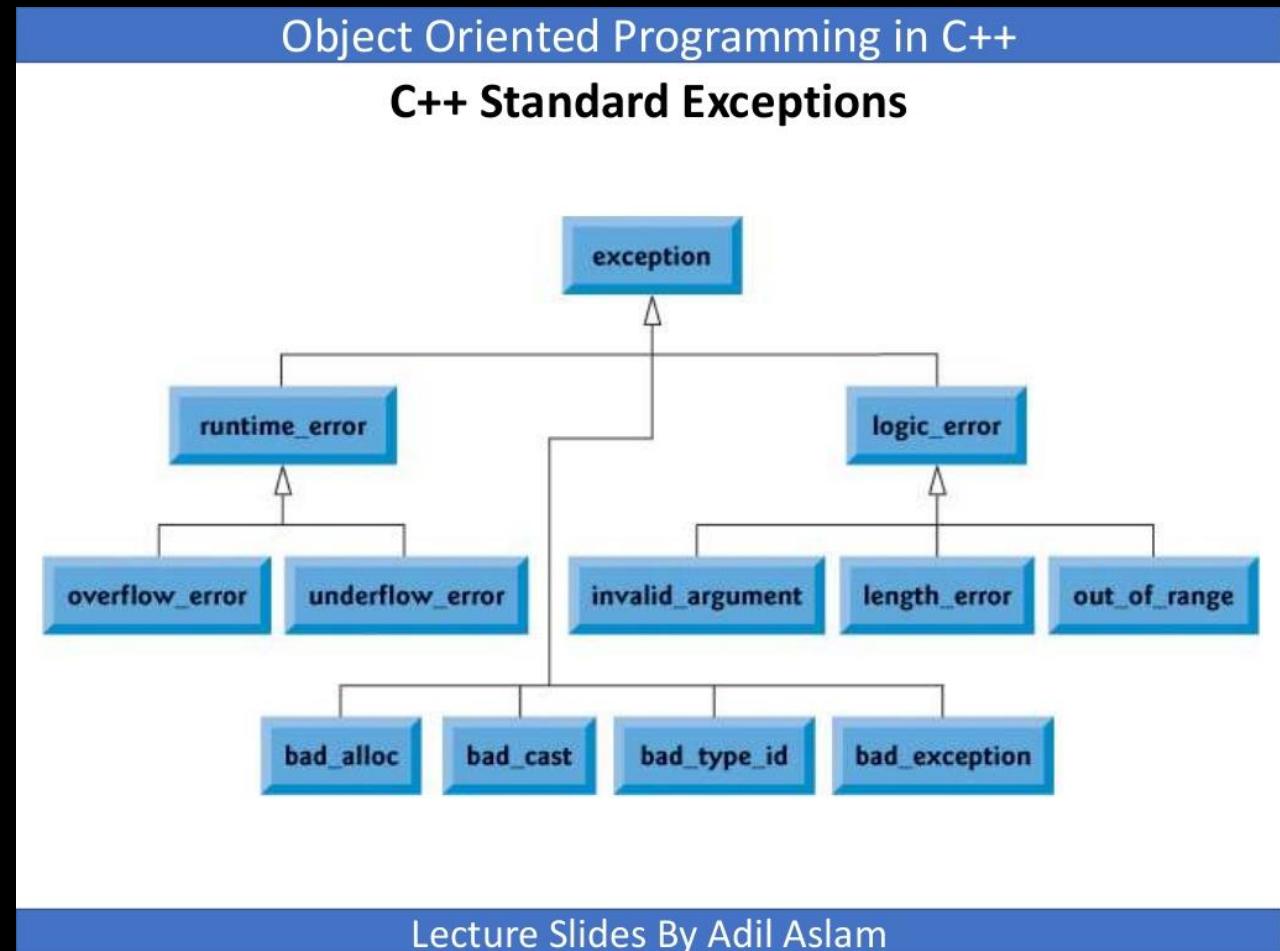
Служебное слово TRY позволяет выделить в любом месте исполняемого текста программы так называемый контролируемый блок:

```
TRY {  
    //ОПЕРАТОРЫ  
    //ОПЕРАТОРЫ  
} CATCH (Тип_исключения1 имя) {  
    //ОПЕРАТОРЫ  
} CATCH (Тип_исключения2 имя) {  
    //ОПЕРАТОРЫ  
}
```

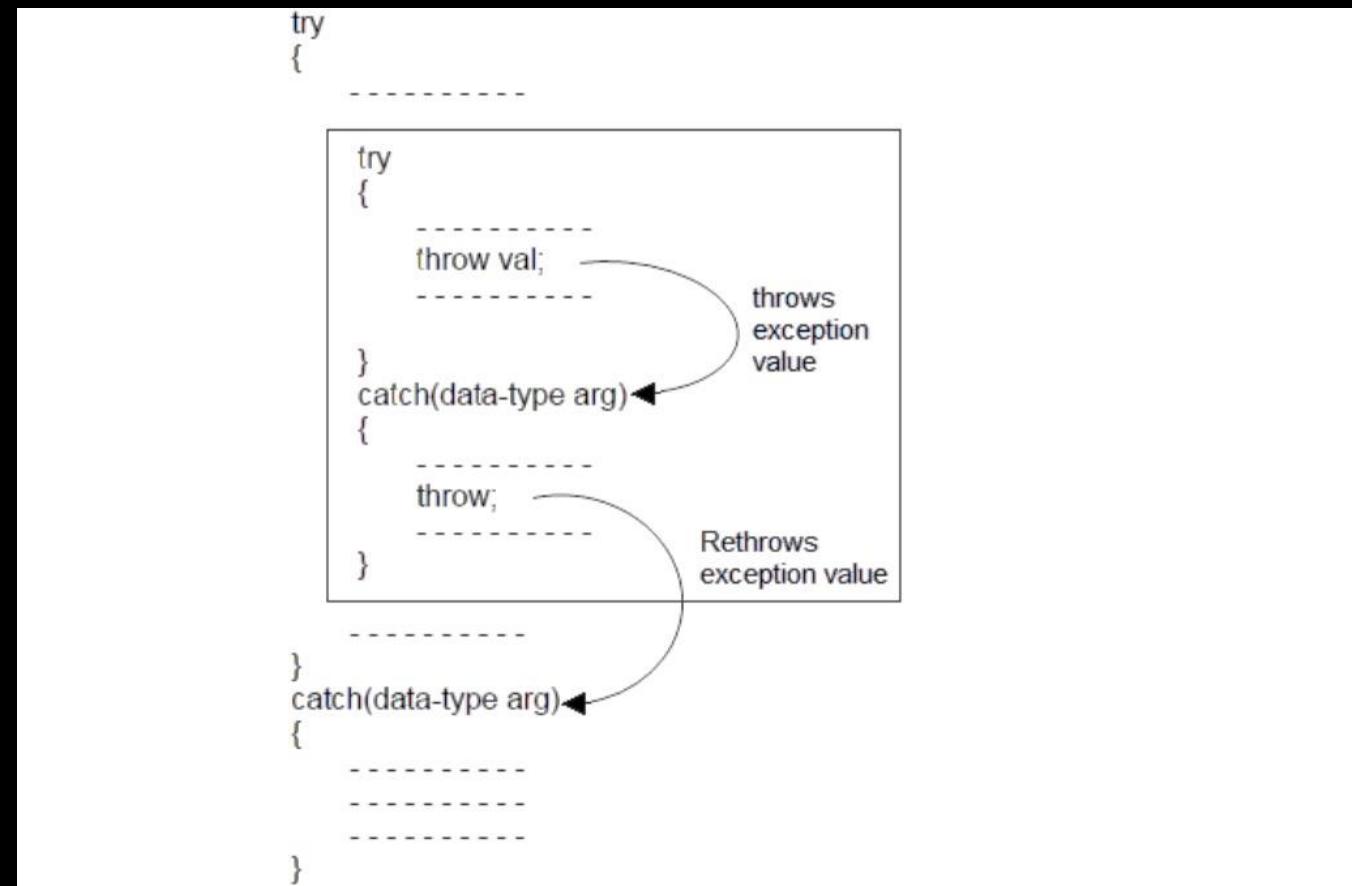
# АРГУМЕНТ ИСКЛЮЧЕНИЯ – ЭТО ВСЕ ЧТО УГОДНО

- C++ ПОЗВОЛЯЕТ СОЗДАВАТЬ ИСКЛЮЧЕНИЯ ЛЮБОГО ТИПА, ХОТЯ ОБЫЧНО РЕКОМЕНДУЕТСЯ СОЗДАВАТЬ ТИПЫ, ПРОИЗВОДНЫЕ ОТ `STD::EXCEPTION`. ИСКЛЮЧЕНИЕ В C++ МОЖЕТ БЫТЬ ПЕРЕХВАЧЕНО ОБРАБОТЧИКОМ `CATCH`, В КОТОРОМ ОПРЕДЕЛЕН ТОТ ЖЕ ТИП, ЧТО И У СОЗДАННОГО ИСКЛЮЧЕНИЯ, ИЛИ ОБРАБОТЧИКОМ, КОТОРЫЙ СПОСОБЕН ПЕРЕХВАТЫВАТЬ ЛЮБОЙ ТИП ИСКЛЮЧЕНИЯ.
- ЕСЛИ СОЗДАННОЕ ИСКЛЮЧЕНИЕ ИМЕЕТ ТИП КЛАССА, У КОТОРОГО ИМЕЕТСЯ ОДИН ИЛИ НЕСКОЛЬКО БАЗОВЫХ КЛАССОВ, ТО ЕГО МОГУТ ПЕРЕХВАТЫВАТЬ ОБРАБОТЧИКИ, КОТОРЫЕ ПРИНЯМИАЮТ БАЗОВЫЕ КЛАССЫ (И ССЫЛКИ НА БАЗОВЫЕ КЛАССЫ) ЭТОГО ТИПА ИСКЛЮЧЕНИЯ.
- ОБРАТИТЕ ВНИМАНИЕ, ЧТО ЕСЛИ ИСКЛЮЧЕНИЕ ПЕРЕХВАТЫВАЕТСЯ ПО ССЫЛКЕ, ТО ОНО ПРИВЯЗЫВАЕТСЯ К САМОМУ ОБЪЕКТУ ИСКЛЮЧЕНИЯ; В ПРОТИВНОМ СЛУЧАЕ ОБРАБАТЫВАЕТСЯ ЕГО КОПИЯ (КАК И В СЛУЧАЕ С АРГУМЕНТАМИ ФУНКЦИИ).

# СТАНДАРТНЫЕ ИСКЛЮЧЕНИЯ C++ #INCLUDE <EXCEPTION>



# ПОВТОРНОЕ «ВОЗБУЖДЕНИЕ» ИСКЛЮЧЕНИЙ



# EXCEPTION PTR

- **STD::CURRENT\_EXCEPTION** — данная функция возвращает EXCEPTION\_PTR. Если мы находимся внутри блока CATCH, то возвращает EXCEPTION\_PTR, который содержит обрабатываемое в данный момент текущим потоком исключение, если вызывать ее вне блока CATCH, то она вернет пустой объект EXCEPTION\_PTR
- **STD::RETHROW\_EXCEPTION** — данная функция бросает исключение, которое содержится в EXCEPTION\_PTR. Если входной параметр не содержит исключений (пустой объект), то результат не определен.
- **STD::MAKE\_EXCEPTION\_PTR** — данная функция, может сконструировать EXCEPTION\_PTR без бросания исключения.

# ПРИМЕР

18\_COMPLEXEXCEPTION

# ВНИМАНИЕ!

- Помни, что в блоке CATCH могут возникать свои исключения!
- Если в блоке CATCH идет высвобождение ресурсов (удаление объектов, закрытие дескрипторов файлов) то их надо самих помещать в еще один вложенный блок TRY/CATCH.

# ПРИМЕР

19\_EXCEPTIONINCATCH

# ОБРАБОТКА ВСЕХ ИСКЛЮЧЕНИЙ

```
TRY  {  
    THROW  CSOMEOTHEREXCEPTION();  
}  
  
CATCH( . . . )  {  
    // CATCH ALL EXCEPTIONS - DANGEROUS!!!  
    // RESPOND (PERHAPS ONLY PARTIALLY) TO THE EXCEPTION,  
    THEN  
    // RE-THROW TO PASS THE EXCEPTION TO SOME OTHER HANDLER  
    // . . .  
    THROW;  
}
```

МОЖНО ЛИ  
ИСПОЛЬЗОВАТЬ  
EXCEPTION КАК  
RETURN «НА  
СТЕРОИДАХ»?

20\_ExFAST

# NOEXCEPT

- В СТАНДАРТЕ ISO C++11 БЫЛ ПРЕДСТАВЛЕН ОПЕРАТОР **NOEXCEPT**.
- **ЕСЛИ МЕТОД С NOEXCEPT ВСЕ ТАКИ СГЕНЕРИРУЕТ ИСКЛЮЧЕНИЕ, ТО ОНО ПЕРЕХВАЧЕНО УЖЕ НЕ БУДЕТ.**

ПРИМЕР

21\_NOEXCEPT

# EXCEPTIONS

## ИТОГО

1. ПОМОГАЕТ СОЗДАТЬ НАДЕЖНУЮ ПРОГРАММУ;
2. ОТДЕЛЯЕТ КОД ОБРАБОТКИ ОШИБОК ОТ ОСНОВНОЙ ЛОГИКИ ПРОГРАММЫ;
3. ОБРАБОТКА ИСКЛЮЧЕНИЙ МОЖЕТ БЫТЬ РЕАЛИЗОВАНА ЗА ПРЕДЕЛАМИ ОСНОВНОГО КОДА ПРОГРАММЫ;
4. СУЩЕСТВУЕТ ВОЗМОЖНОСТЬ ОБРАБАТЫВАТЬ ТОЛЬКО ВЫБРАННЫЕ ТИПЫ ИСКЛЮЧЕНИЙ;
5. ПРОГРАММА, ОБРАБАТЫВАЮЩАЯ ИСКЛЮЧЕНИЯ НЕ ОСТАНОВИТСЯ БЕЗ ОБЪЯСНЕНИЯ ПРИЧИН (НАПРИМЕР, ВЫВОДА НА ЭКРАН ПРИЧИНЫ ВОЗНИKНОVЕНИЯ ИСКЛЮЧЕНИЙ);

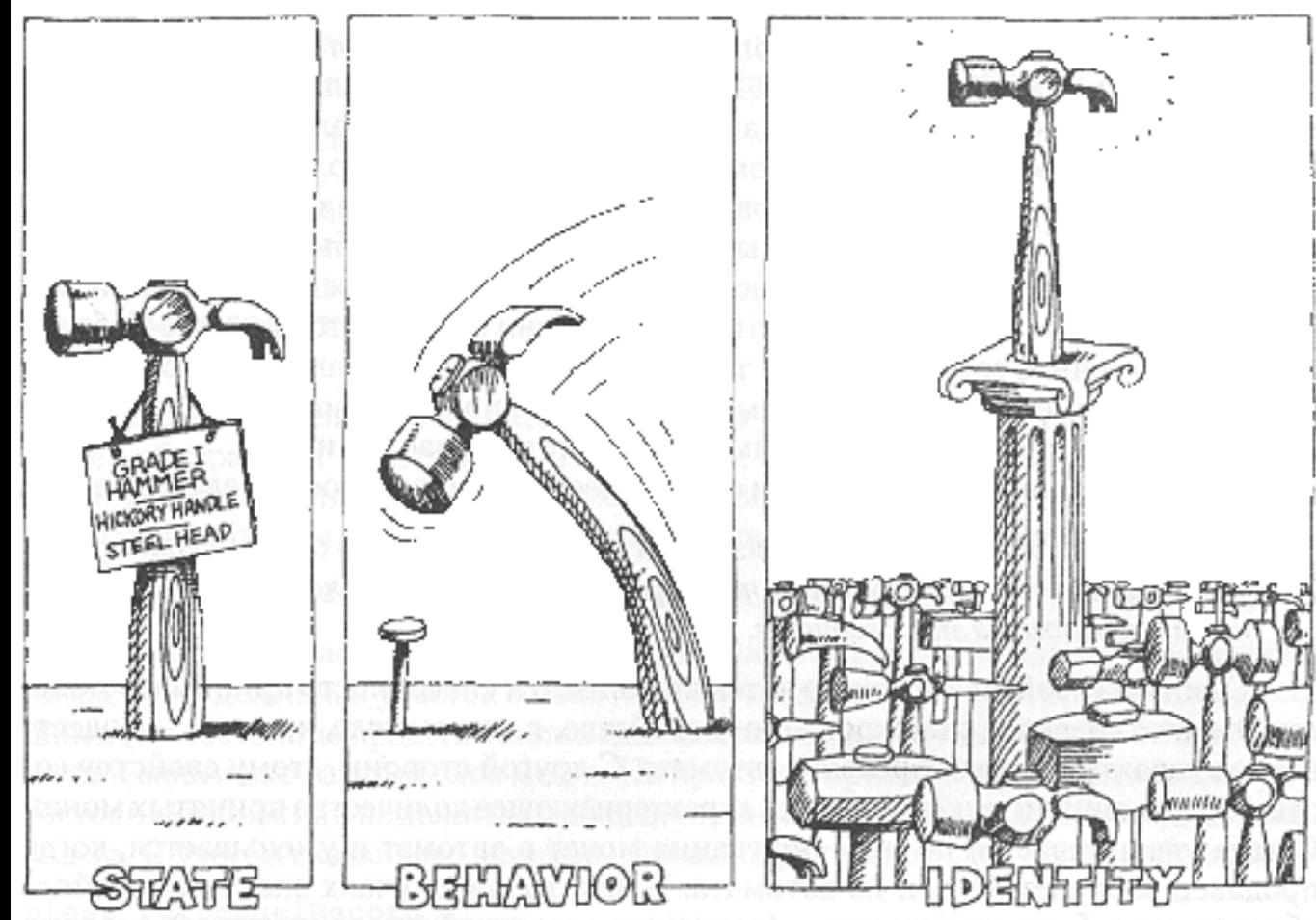


КЛАССЫ И ОБЪЕКТЫ

# ОБЪЕКТ

«ОБЪЕКТ ПРЕДСТАВЛЯЕТ СОБОЙ КОНКРЕТНЫЙ ОПОЗНАВАЕМЫЙ ПРЕДМЕТ, ЕДИНИЦУ ИЛИ СУЩНОСТЬ (РЕАЛЬНУЮ ИЛИ АБСТРАКТНУЮ), ИМЕЮЩУЮ ЧЕТКО ОПРЕДЕЛЕННОЕ ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ В ДАННОЙ ПРЕДМЕТНОЙ ОБЛАСТИ»

SMITH, M. AND TOCKEY, S. 1988. AN INTEGRATED APPROACH TO SOFTWARE REQUIREMENTS DEFINITION USING OBJECTS. SEATTLE, WA: BOEING COMMERCIAL AIRPLANE SUPPORT DIVISION, P.132.



# СВОЙСТВА ОБЪЕКТА

## 1. Состояние

В ЛЮБОЙ МОМЕНТ ВРЕМЕНИ ОБЪЕКТ НАХОДИТСЯ В КАКОМ-ЛИБО СОСТОЯНИИ, КОТОРОЕ МОЖНО ИЗМЕРИТЬ / СРАВНИТЬ / СКОПИРОВАТЬ

## 2. Поведение

ОБЪЕКТ МОЖЕТ РЕАГИРОВАТЬ НА ВНЕШНИЕ СОБЫТИЯ ЛИБО МЕНЯТЬ СВОЕ СОСТОЯНИЕ, ЛИБО СОЗДАВАЯ НОВЫЕ СОБЫТИЯ

## 3. Идентификация

ОБЪЕКТ ВСЕГДА МОЖНО ОТЛИЧИТЬ ОТ ДРУГОГО ОБЪЕКТА

# КЛАСС

## 1. ОПРЕДЕЛЕНИЕ.

КЛАССОМ БУДЕМ НАЗЫВАТЬ ГРУППУ ОБЪЕКТОВ, С ОБЩЕЙ СТРУКТУРОЙ И ПОВЕДЕНИЕМ.

2. Смысл программы на C++ это описание классов!
3. Даже если нужен всего один объект – мы будем описывать класс.

# ОЧЕНЬ ПРОСТОЙ КЛАСС ОБЪЕКТОВ

```
CLASS MyClass
{
    PUBLIC:
        INT NUMBER;
        VOID DOSOMETHING();
    };
}
```

- **CLASS** – КЛЮЧЕВОЕ СЛОВО
- **PUBLIC** – ОБЛАСТЬ ВИДИМОСТИ АТРИБУТОВ И МЕТОДОВ КЛАССА

**INT** NUMBER – АТРИБУТ КЛАССА

**VOID** DOSOMETHING() - МЕТОД КЛАССА



# ПРИМЕРЫ

- 22\_YOURFIRSTCLASS
- 23\_CONSTRUCTORS
- 24\_DEFAULTEDCONSTRUCTORS
- 25\_SETTERSANDGETTERS
- 26\_CLASSACROSSMULTIPLEFILES
- 27\_MANAGINGCLASSOBJECTSTHROUGHPOINTERS
- 28\_DESTRUCTORS
- 29\_ORDEROFCONSTRUCTORDESTRUCTORCALLS
- 30\_THISPOINTER
- 31\_STRUCT
- 32\_SIZEOFCCLASSOBJECTS
- 33\_CONSTMEMBER

# ИНКАПСУЛЯЦИЯ: УПРАВЛЕНИЕ ДОСТУПОМ В С++

ЧЛЕН КЛАССА МОЖЕТ БЫТЬ **ЧАСТНЫМ (PRIVATE)**, **ЗАЩИЩЕННЫМ (PROTECTED)** ИЛИ **ОБЩИМ (PUBLIC)**:

1. **Частный** член класса X могут использовать только функции-члены и друзья класса X.
2. **Защищенный** член класса X могут использовать только функции-члены и друзья класса X, а так же функции-члены и друзья всех производных от X классов (рассмотрим далее).
3. **Общий** член класса можно использовать в любой функции.

Контроль доступа применяется единообразно ко всем именам. На контроль доступа не влияет, какую именно сущность обозначает имя.

Друзья класса объявляются с помощью ключевого слова **FRIEND**. Объявление указывается в описании того класса, к частным свойствам и методам которого нужно получить доступ.

# КОНСТРУКТОР

ЕСЛИ У КЛАССА ЕСТЬ КОНСТРУКТОР, ОН ВЫЗЫВАЕТСЯ ВСЯКИЙ РАЗ ПРИ СОЗДАНИИ ОБЪЕКТА ЭТОГО КЛАССА.

ЕСЛИ У КЛАССА ЕСТЬ ДЕСТРУКТОР, ОН ВЫЗЫВАЕТСЯ ВСЯКИЙ РАЗ, КОГДА УНИЧТОЖАЕТСЯ ОБЪЕКТ ЭТОГО КЛАССА.

ОБЪЕКТ МОЖЕТ СОЗДАВАТЬСЯ КАК:

1. АВТОМАТИЧЕСКИЙ, КОТОРЫЙ СОЗДАЕТСЯ КАЖДЫЙ РАЗ, КОГДА ЕГО ОПИСАНИЕ ВСТРЕЧАЕТСЯ ПРИ ВЫПОЛНЕНИИ ПРОГРАММЫ, И УНИЧТОЖАЕТСЯ ПО ВЫХОДЕ ИЗ БЛОКА, В КОТОРОМ ОН ОПИСАН;
2. СТАТИЧЕСКИЙ, КОТОРЫЙ СОЗДАЕТСЯ ОДИН РАЗ ПРИ ЗАПУСКЕ ПРОГРАММЫ И УНИЧТОЖАЕТСЯ ПРИ ЕЕ ЗАВЕРШЕНИИ;
3. ОБЪЕКТ В СВОБОДНОЙ ПАМЯТИ, КОТОРЫЙ СОЗДАЕТСЯ ОПЕРАЦИЕЙ NEW И УНИЧТОЖАЕТСЯ ОПЕРАЦИЕЙ DELETE;
4. ОБЪЕКТ-ЧЛЕН, КОТОРЫЙ СОЗДАЕТСЯ В ПРОЦЕССЕ СОЗДАНИЯ ДРУГОГО КЛАССА ИЛИ ПРИ СОЗДАНИИ МАССИВА, ЭЛЕМЕНТОМ КОТОРОГО ОН ЯВЛЯЕТСЯ.

СКОЛЬКО РАЗ  
ВЫЗОВЕТСЯ  
КОНСТРУКТОР?

```
STRUCT INTEGER {  
    INT VAL;  
    INTEGER () {  
        VAL = 0;  
        STD::COUT << "DEFAULT CONSTRUCTOR" << STD::ENDL;  
    }  
};  
  
INT MAIN () {  
    INTEGER ARR[3];  
}
```

# ПРАВИЛО ТРЕХ

ПРАВИЛО ТРЁХ (ТАКЖЕ ИЗВЕСТНОЕ КАК «ЗАКОН Большой Тройки» ИЛИ «Большая Тройка») — ПРАВИЛО В C++, ГЛАСЯЩЕЕ, ЧТО ЕСЛИ КЛАСС ИЛИ СТРУКТУРА ОПРЕДЕЛЯЕТ ОДИН ИЗ СЛЕДУЮЩИХ МЕТОДОВ, ТО ОНИ ДОЛЖНЫ ЯВНЫМ ОБРАЗОМ ОПРЕДЕЛИТЬ ВСЕ ТРИ МЕТОДА:

- ДЕСТРУКТОР
- Конструктор копирования
- Оператор присваивания копированием

# КОНСТАНТНЫЕ ФУНКЦИИ

```
STRUCT A {  
    INT X;  
    VOID F(INT A) CONST {  
        X = A; // -- НЕ РАБОТАЕТ  
    }  
};
```

# КАК ПРИДУМАТЬ КЛАСС?

АБСТРАКЦИЯ ВЫДЕЛЯЕТ СУЩЕСТВЕННЫЕ ХАРАКТЕРИСТИКИ НЕКОТОРОГО ОБЪЕКТА, ОТЛИЧАЮЩИЕ ЕГО ОТ ВСЕХ ДРУГИХ ВИДОВ ОБЪЕКТОВ И, ТАКИМ ОБРАЗОМ, ЧЕТКО ОПРЕДЕЛЯЕТ ЕГО КОНЦЕПТУАЛЬНЫЕ ГРАНИЦЫ С ТОЧКИ ЗРЕНИЯ НАБЛЮДАТЕЛЯ.

# АБСТРАКЦИЯ СУЩНОСТИ

- ОБЪЕКТ ПРЕДСТАВЛЯЕТ СОБОЙ ПОЛЕЗНУЮ МОДЕЛЬ НЕКОЙ СУЩНОСТИ В ПРЕДМЕТНОЙ ОБЛАСТИ

# АБСТРАКЦИЯ ПОВЕДЕНИЯ

Объект состоит из  
обобщенного множества  
операций

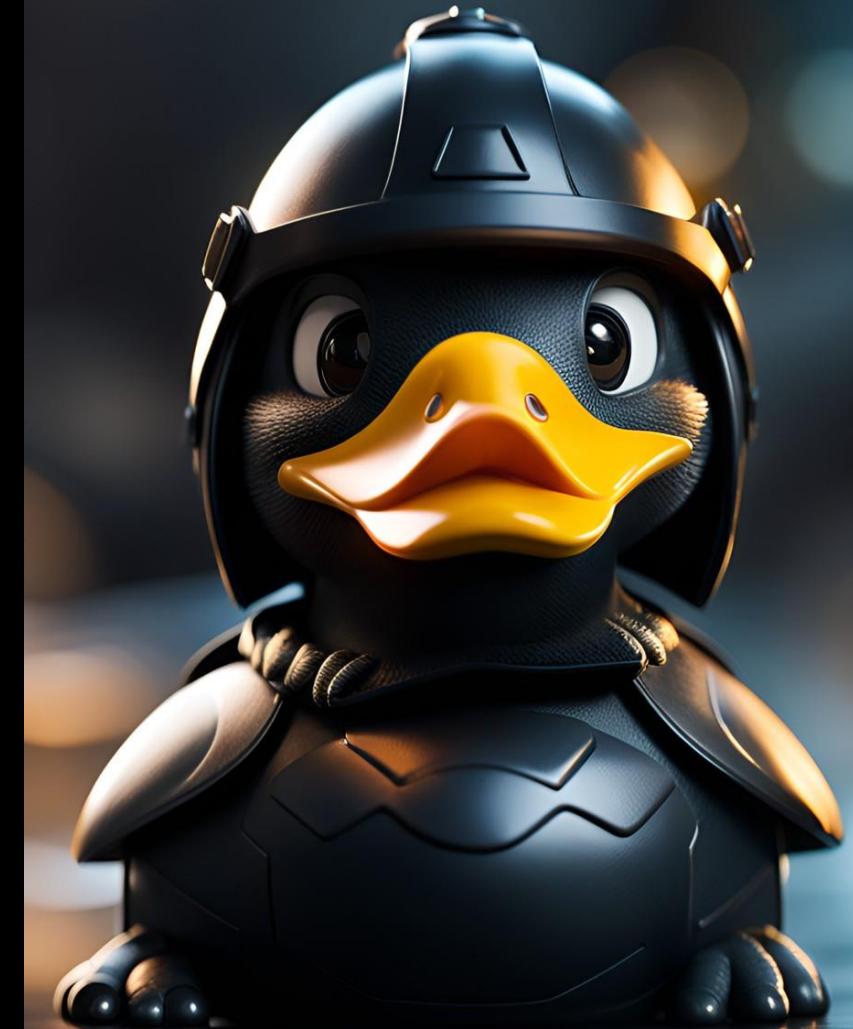
# АБСТРАКЦИЯ ВИРТУАЛЬНОЙ МАШИНЫ

ОБЪЕКТ ГРУППИРУЕТ ОПЕРАЦИИ, КОТОРЫЕ ЛИБО ВМЕСТЕ ИСПОЛЬЗУЮТСЯ БОЛЕЕ ВЫСОКИМ УРОВНЕМ УПРАВЛЕНИЯ, ЛИБО САМИ ИСПОЛЬЗУЮТ НЕКОТОРЫЙ НАБОР ОПЕРАЦИЙ БОЛЕЕ НИЗКОГО УРОВНЯ

# ПРОИЗВОЛЬНАЯ АБСТРАКЦИЯ

ОБЪЕКТ ВКЛЮЧАЕТ В СЕБЯ НАБОР ОПЕРАЦИЙ, НЕ ИМЕЮЩИХ ДРУГ С  
ДРУГОМ НИЧЕГО ОБЩЕГО

# ЛАБОРАТОРНАЯ РАБОТА №2



# СПАСИБО!

НА СЕГОДНЯ ВСЕ