Last updated: 2019-05-01
vadimov@i.ua

Практическое занятие 12 (семестр2).

Задача 12.1. Напишите родовую функцию **min()**, возвращающую меньший из двух своих аргументов. Например, версия функции **min(3, 4)** должна возвратить **3**, а версия **min('c', 'a') - a.** Продемонстрируйте работу функции с помощью программы.

Задача 12.2. Функция **find()** ищет объект в массиве. Она возвращает либо индекс найденного объекта (если его удалось найти), либо -1, если заданный объект не найден. Ниже представлен прототип конкретной версии функции **flnd()**:

```
int find(int object, int *list, int size) {
    // Параметр size задает количество элементов массива.
}
```

Переделайте функцию **find()** в родовую функцию и продемонстрируйте ваше решение в программе.

Задача 12.3. Создайте и продемонстрируйте родовые классы, реализующие очередь и стек. Подсказка: вот начало родового класса, реализующего стек. Посмотрите Example 2.6 из Unit 2 и закончите написание кода.

```
template <class StackType> class Stack {
    StackType stck[SIZE];     // holds the stack
    int tos;                  // index of top of stack
public:
    void init();              // initialize stack
    void push(StackType ch);  // push object on stack
    StackType pop();          // pop object from stack
};
template <class StackType>
void Stack<StackType> :: push(StackType obj) {
...
```

Задача 12.4. Создайте родовой класс **Input**, который при вызове конструктора делает следующее:

- выводит на экран строку-приглашение,
- получает данные от пользователя,
- повторно выводит на экран строку-приглашение, если вводимые данные не соответствуют заданному диапазону.

Объекты типа **Input** должны объявляться следующим образом:

```
Input obj("prompt message> ", min_value, max_value)
```

Здесь prompt message - это сообщение, появляющееся на экране в качестве приглашения для ввода. Минимальное и максимальное допустимые значения задаются с помощью параметров min_value и max_value соответственно. (Тип данных, вводимых пользователем, будет тем же самым, что и тип значений min_value и max_value.)

Задача 12.5. Здесь представлен каркас функции **divide()**:

```
double divide (double a, double b) {
    // add error handling
    return a/b;
}
```
Эта функция возвращает результат деления a на b. Добавьте в функцию код для обработки исключительных ситуаций, а конкретно предусмотрите обработку ошибки деления на ноль. Покажите, что ваша программа работает.

Задача 12.6. Создайте родовую функцию, возвращающую значение элемента, который чаще всего встречается в массиве.

Задача 12.7. Создайте родовую функцию, возвращающую сумму значений элементов массива.

Задача 12.8. Создайте родовой класс для "пузырьковой" сортировки (или используйте любой другой известный вам алгоритм сортировки).

Задача 12.9. Измените класс **Stack** так, чтобы в стеке можно было хранить пары объектов разных типов.

Задача 12.10. Еще раз измените класс **Stack** так, чтобы переполнение и, наоборот, опустошение стека обрабатывались как исключительные ситуации.

Задача 12.11. Просмотрите документацию на ваш компилятор. Проверьте, поддерживает ли он функции **terminate()** и **unexpected()**. Как правило, эти функции можно конфигурировать так, чтобы из них вы могли вызвать любую необходимую вам функцию. Если в случае с вашим компилятором это так, постарайтесь создать собственный набор функций завершения программы, который обеспечил бы возможность обработки необрабатываемых до этого исключительных ситуаций.

Задача 12.12. Вопрос для размышления: в чем, по вашему мнению, при неудачной попытке выделения памяти преимущество возбуждения исключительной ситуации оператором new по сравнению с возвращением нуля?

Задача 12.13. В Unit 2 "Introducing Function Overloading" были созданы перегруженные версии функции **abs()**. Усовершенствуйте решение, создав родовую функцию **abs()**, которая возвращала бы абсолютную величину любого численного объекта.

Задача 12.14. В Unit5 "Bounded Array" рассматривался простейший способ реализации безопасного массива, в котором для доступа к элементам массива использовались функции **get()** и **put()**. Перегрузка оператора **[]** позволяет создать такой массив гораздо проще. В практическом задании для Unit7 вам поручалось для создания безопасного массива реализовать в функции **operator[]()** контроль границ. Кроме этого, функция **operator[]()** должна возвращать ссылку на индексируемый элемент. В представленном ниже фрагменте сделана попытка добавить контроль границ массива, что позволяет при нарушении границ генерировать соответствующую ошибку.

Подсказка: Вспомните, что безопасный массив - это массив, который инкапсулирован в классе, и при этом класс обеспечивает контроль границ массива. Благодаря перегрузке оператора **[],** работать с безопасным массивом можно так же, как с обычным.

```
// A safe array example:
...
class arraytype {
    ...
};
// Provide range checking for arraytype .
int &arraytype::operator [](int i) {
    if (...) {
        cout << "Index value of " << i << " is out of bounds.\n";
        ...
    }
    ...
}
int main() {
    arraytype obj;
    // this generates a run - time error because SIZE+100 is out of range:
    obj[SIZE+100] = 99; // error
    return 0;
}
```

Итак, у вас есть уже класс с безопасным массивом. Переделайте его в родовой класс с безопасным массивом.