Задания на самостоятельное выполнение

1 Задание №1

Разработать и реализовать программный модуль, обеспечивающий работу со стеком ненулевых элементов целого типа.

1.1 Описание структуры данных

```
struct stack_element{
  int element;
  struct stack_element *prev;
};

struct _stack{
  struct stack_element *top;
  int buffer;
};

typedef struct _stack *stack;
```

Структура `struct stack_element` описывает структуру данных отдельного элемента стека. Описание полей данной структуры:

- `int element` — переменная, в которой храниться значение элемента стека;
- `struct stack_element *prev` — указатель на предыдущий элемент стека; если предыдущий элемент отсутствует, то данный указатель имеет значение `NULL`.

Структура `struct _stack` описывает стек в целом. Описание полей данной структуры:

- `struct stack_element *top` — указатель на вершину стека; если в стеке нет элементов, то значение этого указателя равно `NULL`;

- `int buffer` — переменная, в которую помещается значение выталкиваемого элемента; необходимость наличия такой переменной связана с тем, что операция выталкивания элемента из стека должна не только возвращать значение этого элемента, но и освобождать память, динамически выделенную под выталкиваемый элемент.

Операция `typedef struct _stack *stack` обеспечивает возможность использовать имя `stack`, как имя типа для "стековых" переменных.

1.2 Описание операций работы со стеком

Для работы со стеком должны быть реализованы следующие операции:

- создать стек;
- поместить элемент в стек;
- вытолкнуть элемент из стека;
- удалить стек.

1.2.1 Создать стек

```
stack create_stack();
```

Параметры и возвращаемые значения:

- функция возвращает значение адреса динамически выделенной памяти, если не было ошибок при создании стека. Иначе, функция возвращает значение NULL.

1.2.2 Поместить элемент в стек

```
int push(stack _stack, int _element);
```

Параметры и возвращаемые значения:

- `_stack` — указатель на стек;
- `int _element` — помещаемое в стек значение;
- функция возвращает значение -1 в случае ошибки и 0 в случае нормального завершения.

1.2.3 Вытолкнуть элемента из стека

```
int pop(stack _stack);
```

Параметры и возвращаемые значения:

- `_stack` — "стековая" переменная;
- функция возвращает значение поля `buffer`. Если стек пуст, то функция возвращает значение 0.

Таким образом, данная функция должна обеспечить сохранение выталкиваемого из стека значения в поле `buffer`, и освободить память, выделенную ранее под выталкиваемый элемент.

1.2.4 Удалить стек

```
void delete_stack(stack _stack);
```

Параметры и возвращаемые значения:

- `_stack` — удаляемая "стековая" переменная;
- функция освобождает динамически выделенную память.

1.3 Требования к выполнению работы

- Описания структуры и функций должны быть помещены в файл `stack.h`.
- Реализация функций должна быть представлена в файле `stack.c`.
- Тестовая программа должна быть представлена в файле `test01.c`.
- Тестовая программа должна обеспечить проверку всех возможных ситуаций работы функций.

2 Задание №2

Разработать и реализовать классы, обеспечивающие работу со стеком ненулевых элементов целого типа.

За основу взять реализацию модуля, обеспечивающего работу со стеком ненулевых элементов целого типа, разработанного при выполнении задания №1.

В результате выполнения данного задания должны быть разработаны, реализованы и протестированы следующие классы:

- **IntStackElement** — класс, обеспечивающий работу с отдельным элементом стека (аналогичен структуре **struct stack_element** из задания №1);

- **IntStack** — класс, обеспечивающий работу со стеком ненулевых элементов целого типа (аналогичен структуре **struct _stack** из задания №1.

Вместо операций (функций) **create_stack()** и **delete_stack()**, реализованных в задании №1, в рамках класса **IntStack** должны быть предусмотрены конструктор и деструктор соответственно, а именно:

- **IntStack()** — конструктор, обеспечивающий инициализацию элементов класса **IntStack**;
- **~IntStack()** — деструктор, обеспечивающий удаление стека.

Вместо операций (функций) **push()** и **pop()**, реализованных в задании №1, в рамках класса **IntStack** должны быть предусмотрены следующие методы:

- **void Push(_element)** — поместить элемент со значением `_element` в стек;
- **int Pop()** — удалить элемент из стека и его значение вернуть в качестве результата.

Для динамического выделения памяти и освобождения динамически выделенной памяти использовать операции языка программирования C++ **new** и **delete** соответственно.

Пример использования операций **new** и **delete**:

```
//===== Файл example.hpp =====
class Example{
    int element;
public:
    Example();
    void SetElement(int _element);
    int GetElement();
};

//===== Файл example.cpp =====
#include "example.hpp"
Example::Example(){
    element = 777;
}

void Example::SetElement(int _element){
    element = _element;
}

int Example::GetElement(){
    return element;
```

```
}

//===== Файл test.cpp =====
#include<stdio.h>
#include<stdlib.h>
#include "example.hpp"
int main(){
    Example example01, example02;
    Example * example_ptr01, * example_ptr02;
    printf("example01 == %d; example02 == %d\n",
            example01.GetElement(), example02.GetElement());
    example01.SetElement(123);
    example02.SetElement(321);
    printf("example01 == %d; example02 == %d\n",
           example01.GetElement(), example02.GetElement());
    example_ptr01 = (Example*)calloc(1, sizeof(Example));
    printf("example_ptr01->GetElement() = %d\n",
            example_ptr01->GetElement());
    example_ptr02 = new Example;
    printf("example_ptr02->GetElement() = %d\n",
            example_ptr02->GetElement());
}
```

2.1 Требования к выполнению работы

- Описания структуры и функций должны быть помещены в файл `stack.hpp`.
- Реализация функций должна быть представлена в файле `stack.cpp`.
- Тестовая программа должна быть представлена в файле `test02.cpp`.
- Тестовая программа должна обеспечить проверку всех возможных ситуаций работы функций.

3 Задание №3

В предыдущем задании были реализованы классы и методы, обеспечивающие работу со стеком целых **НЕНУЛЕВЫХ** элементов.

В данной работе необходимо изменить класс **IntStack** таким образом, чтобы была обеспечена работа со стеком любых целых чисел, включая значение `0`.

Вопросы, которые возникают при выполнении данного задания:

- каким образом обеспечить возможность определения корректности выполнения метода `Pop()`? Например, при выполнении метода `Pop()` может быть ситуация, когда в стеке отсутствуют элементы.
  В предыдущем задании метод Pop() возвращал значение 0 в случаях

невозможности выполнения данной операции. Это было приемлемо, поскольку стек формировался только для ненулевых элементов и значение 0 могло быть использовано в качестве результата выполнения метода `Pop()`.

- каким образом обеспечить возможность корректности выполнения метода `Push(...)`? Например, при выполнении метода `Push(...)` возможна ситуация, когда нет оперативной памяти для размещения очередного элемента стека.

Для решения этих проблем, в данном задании необходимо воспользоваться механизмом **исключительных ситуаций** языка программирования `C++` (см. [1], стр. 222–230).

В соответствии с этим, в классе **IntStack** необходимо предусмотреть следующие типы исключительных ситуаций:

- `Empty` — исключительная ситуация, возникающая при выполнении метода `Pop()` по отношению к стеку, в котором нет элементов;
- `NoMemory` — исключительная ситуация, возникающая при выполнении метода `Push(...)` в случае отсутствия свободной памяти под помещаемый в стек элемент.

3.1 Обработка исключительных ситуаций

Для обработки исключительных ситуаций используются следующие операции:

- **try** — выполнить операцию с отслеживанием возможных исключительных операций;
- **catch** — обработать исключительную ситуацию;
- **throw** — породить исключительную ситуацию.

3.1.1 Общая схема обработки исключительных ситуаций.

```
class Test{
  ...
public:
  class ExeptionClass{};
  class OtherExeptionClass{};
  /* Типы возможных исключительных ситуаций */
  ...
  void SomeMethod() throw(ExeptionClass, OtherExeptionClass);
  /* Метод, который может породить исключительные ситуации
     ExeptionClass, OtherExeptionClass */
  ...
};
```

```
Test::SomeMethod() throw(ExeptionClass, OtherExeptionClass){
  ...
  if(...){
    throw ExeptionClass();
    /* Порождение исключительной ситуации
       ExeptionClass */
  }
  ...
  if(...){
    throw OtherExeptionClass();
    /* Порождение исключительной ситуации
       OtherExeptionClass */
  }
  ...
}
...
int main(){
  Test atest;
  ...
  try{
    atest.SomeMethod();
    /* Вызов метода SomeMethod() объекта atest
       с контролем возможных исключительных ситуаций */
  }
  catch(Test::ExeptionClass){
    ...
    /* Обработка исключительной ситуации типа ExeptionClass
       класса Test */
  }

  catch(Test::OtherExeptionClass){
    ...
    /* Обработка исключительной ситуации типа OtherExeptionClass
       класса Test */
  }
  ...
}
```

3.1.2 Исключительные ситуации при выполнении операции new

При выполнении операции **new** порождается исключительная ситуация **bad_alloc** в том случае, когда отсутствует объём памяти необходимого размера.

Для обработки данной исключительной ситуации в файл с исходным кодом должен быть включён заголовочный файл с именем **new**. Пример:

```
#include <new>
...
void f(int n){
  int *iptr;
  ...
  try{
    iptr = new int[n];
  }
  catch(std::bad_alloc){
    ...
    /* Обработка исключительной ситуации */
  }
  ...
}
```

3.2 Требования к выполнению работы

- Описания классов должны быть помещены в файл `stack.hpp`.
- Реализация методов классов должна быть представлена в файле `stack.cpp`.
- Тестовая программа должна быть представлена в файле `test03.cpp`.
- Тестовая программа должна обеспечить проверку всех возможных ситуаций работы с объектами типа **IntStack**.

4 Задание №4.

В данной работе необходимо изменить класс **IntStack** таким образом, чтобы при работе с объектами этого класса были доступны следующие операции:

- операция `<<` обеспечивает помещение целочисленного элемента в стек:
  `stack << intval`
  где

  `stack` — объект класса **IntStack**;

  `intval` — любое целочисленное значение.

  Результат выполнения данной операции должен быть тем же, что и результат выполнения операции `stack.Push(intval)`.
  Если при выполнении данной операции определяется, что недостаточно памяти для помещения в стек очередного элемента, то данная операция порождает исключительную ситуацию **NoMemory**.

- операция `>>` обеспечивает выталкивание элемента из стека:
  `stack >> intvar`
  где

  `stack` — объект класса **IntStack**.

  `intvar` — целочисленная переменная.

  В результате выполнения этой операции значением переменной `intvar` становится значение элемента, выталкиваемого из стека. Результат выполнения данной операции должен быть аналогичен выполнению операции:

  ```
  intvar = stack.Pop()
  ```

  Если данная операция выполняется по отношению к пустому стеку, то порождается исключительная ситуация **Empty**.

- операция `==` обеспечивает сравнение на равенство двух объектов класса **IntStack**:
  `stack01 == stack02` где

  `stack01` и `stack02` — сравниваемые объекты класса **IntStack**.

  Пример 1.
  Пусть объект `stack01` класса **IntStack** содержит следующие значения (вершина стека — крайний правый элемент): `{1, 2, 3, 4, 5}`, а объект `stack02`, относящийся к этому же классу, содержит следующие значения: `{15, 17, 25}`. В этом случае, результатом выполнения операции `stack01 == stack02` будет значение `0`.
  Пример 2.
  Пусть объект `stack01` класса **IntStack** содержит следующие значения (вершина стека — крайний правый элемент): `{15, 17, 25}`, и объект `stack02`, относящийся к этому же классу, содержит такие же значения: `{15, 17, 25}`. В этом случае, результатом выполнения операции `stack01 == stack02` будет значение `1`.

Для выполнения задания необходимо воспользоваться механизмом **перегрузки (переопределения)** операций языка программирования `C++` (см. [1], стр. 58, 77–78, 189–191).

4.1 Требования к выполнению работы

- Описания классов должны быть помещены в файл `stack.hpp`.
- Реализация методов классов должна быть представлена в файле `stack.cpp`.
- Тестовая программа должна быть представлена в файле `test04.cpp`.
- Тестовая программа должна обеспечить проверку всех возможных ситуаций работы с объектами типа **IntStack**.

5 Задание №5.

В данной работе необходимо изменить класс **IntStack** таким образом, чтобы при работе с объектами этого класса были доступны следующие операции:

- операция инициализации стека при его объявлении значением другого стека:

  ```
  IntStack stack02 = stack01
  ```

  или (что то же самое):

  ```
  IntStack stack02(stack01)
  ```

  где

  `stack02` — создаваемый объект класса **IntStack**;

  `stack01` — созданный ранее и, возможно, непустой стек.

  В результате выполнения данной операции, объект `stack02` должен иметь тот же состав элементов, что и объект `stack01`.
  Если при выполнении данной операции определяется, что недостаточно памяти для формирования объекта `stack02`, то данная операция порождает исключительную ситуацию **NoMemory** и, в качестве результата, создаёт незаполненный (пустой) стек `stack02`.
  Пример.

  ```
  IntStack stack01;
  stack01 << 13;
  stack01 << 25;

  IntStack stack02 = stack01;
  IntStack stack03(stack01);
  ```

  В результате выполнения указанных операций должны быть созданы следующие объекты:

  – `stack02`;

  – `stack03`.

  Эти объекты, после выполнения указанных операций, должны содержать следующий набор элементов: `{13, 25}`.

- операция `=` обеспечивает присваивание значения одного объекта класса **IntStack** другому объекту этого же класса:

  ```
  stack02 = stack01
  ```

  где `stack01` и `stack02` — объекты класса **IntStack**.
  В результате выполнения этой операции набор и значения элементов объекта `stack02` должны быть такими же, что и у объекта `stack01`, независимо от состояния объекта `stack02` до выполнения этой операции.
  Если при выполнении данной операции определяется, что недостаточно памяти для формирования объекта `stack02`, то данная операция порождает исключительную ситуацию **NoMemory** и, в качестве результата, возвращает незаполненный (пустой) стек `stack02`.
  Пример.
  Пусть объекты класса **IntStack** имеют следующие значения:

  `stack01`: `{1, 3, 5, 7, 9, 11}`;

  `stack02`: `{13, 25}`.

  В результате выполнения операции:

  ```
  stack01 = stack02
  ```

  набор и значения элементов объектов `stack01` и `stack02` должны быть одинаковы: `{13, 25}`.

- операции `<<` и `>>` должны быть реализованы таким образом, чтобы могли обеспечить множественное помещение значений в стек и множественное выталкивание значений из стека соответственно:

  – `stack << intval01 << intval02 << ... << intvalN`
  где

  `stack` — объект класса **IntStack**;

  `intval01, intval02, ..., intvalN` — любые целочисленные значения.

  Пример.
  Пусть объект `stack` класса **IntStack** содержит следующие значения (вершина стека — крайний правый элемент): `{1, 3}`. Тогда, после выполнения операции:

  ```
  stack << 13 << 25;
  ```

объект `stack` должен содержать следующий набор значений: `{1, 3, 13, 25}`.
Если при выполнении данной операции определяется, что недостаточно памяти для помещения в стек очередного элемента, то порождается исключительная ситуация **NoMemory** и стек, в результате, должен содержать те элементы, которые были в него помещены к моменту возникновения этой исключительной ситуации.

– `stack >> intvar01 >> intvar02 >> ... >> intvarN`
где

`stack` — объект класса **IntStack**;

`intval01, intval02, ..., intvalN` — целочисленные переменные.

Пример.
Пусть переменные `var01` и `var02` являются целочисленными переменными, и объект `stack` класса **IntStack** содержит следующие значения (вершина стека — крайний правый элемент): `{1, 3, 13, 25}`.
Тогда, после выполнения операции:

```
stack >> var01 >> var02;
```

объект `stack` должен содержать следующий набор значений: `{1, 3}`, а переменные `var01` и `var02` должны содержать значения 25 и 13 соответственно.
Если при выполнении данной операции определяется, что стек пуст, то порождается исключительная ситуация **Empty**, и у переменных, не получивших к этому моменту значений из стека, должны остаться те значения, которые они имели до выполнения рассматриваемой операции.

Для выполнения задания необходимо использовать такие средства системы программирования C++, как **конструкторы копирования**, **указатель this** и **перегрузка (переопределение) операции присваивания** (см. [1], стр. 181–185, 191–192).

5.1 Требования к выполнению работы

- Описания классов должны быть помещены в файл `stack.hpp`.
- Реализация методов классов должна быть представлена в файле `stack.cpp`.
- Тестовая программа должна быть представлена в файле `test05.cpp` .
- Тестовая программа должна обеспечить проверку всех возможных ситуаций работы с объектами типа **IntStack**.

6 Задание №6.

Необходимо разработать класс **PrintedIntStack**, являющийся наследником класса **IntStack** и предоставляющий операцию (метод) печати стека **PrintStack**.

В качестве параметров данная операция должна принимать строку символов, содержащую название стека, для которого выполняется данная операция. Название стека задаётся пользователем при вызове операции **PrintStack**.

Формат вывода стека на печать:

```
<название стека>
!   <Z1>!
!   <Z2>!
!.......!
!   <ZN>!
+-------+
```

где:

- `<название стека>` — значение строки символов, переданной в качестве параметра в метод **PrintStack**; если в качестве этого параметра передана пустая строка или адрес начала этой строки равен `0`, то печать названия стека не производится.
- `<Z1>` — значение элемента, находящегося на вершине стека (значение последнего помещённого в стек элемента);
- `<ZN>` — значение элемента, находящегося на "дне" стека (значение первого помещённого в стек элемента.

Ширина поля, выделенного под значения — семь символов.

Если стек не содежит элементов, то при вызове метода **PrintStack** такого стека порождается исключительная ситуация **Empty** и, соответственно, метод **PrintStack** ничего не выводит на экран.

Описание класса **PintedIntStack** должно быть приведено в файле `stack.hpp`, а реализация метода **PrintStack** должна быть приведена в файле `stack.cpp`. За основу берутся файлы, подготовленные при выполнении задания №5.

Тестовая программ должна содержаться в файле `test06.cpp`.

7 Задание №7

В данной работе необходимо создать класс **Stack**, обладающий набором методов, аналогичных методам класса **IntStack**, реализованного в рамках задания №5, но, в отличие от ранее реализованного класса **IntStack**, класс **Stack** должен обеспечить возможность работы с объектами типа стек, обладающими следующими характеристиками:

- каждый отдельный стек должен содержать элементы одного типа;

- тип элементов отдельного стека **не обязательно** является целочисленным.

Например, данный класс должен позволять создавать такие объекты, как стек целых чисел, стек вещественных чисел, стек символов (литер) и т.п.

Для выполнения задания необходимо использовать такие средства системы программирования C++, как **шаблоны классов** и, возможно, **шаблоны функций** (см. [1], глава 6 "Шаблоны классов", стр. 211; раздел "Шаблоны функций", стр. 85).

7.1 Требования к выполнению работы

- Описания классов и реализация методов классов должны быть помещены в файл `stack.hpp`.
- Исключительные ситуации **Empty** и **NoMemory** должны быть вынесены в отдельный класс **Exception**:

```
class Exception{
  public:
      class Empty{};
      class NoMemory{};
};
```

- Тестовая программа должна быть представлена в файле `test07.cpp`
- Тестовая программа должна обеспечить проверку всех возможных ситуаций работы с объектами типа **Stack** такими, как:
  - стек целых чисел (`int`);
  - стек вещественных чисел (`float`);
  - стек символов (`char`).
- Тестовая программа должна обеспечить вывод на экран "трассировки" её выполнения, а именно, на экран должны выводится действия, которые выполняются, и, после каждого действия, должно выводится текущее состояние (содержимое) соответствующего стека.

8 Задание №8

В данной работе необходимо в класс **Stack**, реализованный в рамках задания №7, добавить операцию (метод) **PrintStack**, обеспечивающий печать стека.

В качестве параметров данная операция должна принимать строку символов, содержащую название стека, для которого выполняется данная операция. Название стека задаётся пользователем при вызове операции **PrintStack**.

Формат вывода стека на печать:

```
<название стека>
!   <Z1>!
!   <Z2>!
!.......!
!   <ZN>!
+-------+
```

где:

- `<название стека>` — значение строки символов, переданной в качестве параметра в метод **PrintStack**; если в качестве этого параметра передана пустая строка, то печать названия стека не производится.
- `<Z1>` — значение элемента, находящегося на вершине стека (значение последнего помещённого в стек элемента);
- `<ZN>` — значение элемента, находящегося на "дне" стека (значение первого помещённого в стек элемента.

Ширина поля, выделенного под значения — семь символов.

Если стек не содежит элементов, то при вызове метода **PrintStack** такого стека порождается исключительная ситуация **Empty** и, соответственно, метод **PrintStack** ничего не выводит на экран.

Для выполнения задания необходимо использовать такие стандартные классы системы программирования `C++`, как **потоковые классы** и **строки** (см. [1], глава 10 "Потоковые классы", стр. 265; глава 11 "Строки", стр. 286).

8.1 Требования к выполнению работы

- Описания классов и реализация методов классов должны быть помещены в файл `stack.hpp`.
- Исключительные ситуации **Empty** и **NoMemory** должны быть вынесены в отдельный класс **Exception**:

```
class Exception{
  public:
      class Empty{};
      class NoMemory{};
};
```

- Тестовая программа должна быть представлена в файле `test08.cpp`
- Тестовая программа должна обеспечить проверку всех возможных ситуаций работы с объектами типа **Stack** такими, как:
  - стек целых чисел (`int`);
  - стек вещественных чисел (`float`);

– стек строк символов (`string`).

- Тестовая программа должна обеспечить вывод на экран “трассировки” её выполнения, а именно, на экран должны выводится действия, которые выполняются, и, после каждого действия, должно выводится текущее состояние (содержимое) соответствующего стека.

Список литературы

[1] Т.А. Павловская, C/C++. Программирование на языке высокого уровня, СПб., “Питер”, 2013.