Семинар 3. Стековая машина

С. А. Шершаков

24 сентября 2018 г.

Ред. 1.4 от 24.09.2018 г.

В документе представлен 3-й цикл заданий для самостоятельного выполнения студентами курса «Алгоритмы и структуры данных» и методические рекомендации по их выполнению. Цикл посвящен изучению АТД «стек» и области его применения — в частности, стековым машинам, стековым языкам и форме записи выражений в виде прямой и обратной польской нотаций.

1 Требования, цели и ожидаемые результаты

Чтобы перейти непосредственно к шагам, необходимым для выполнения задания, обратитесь к разделу 4.

1.1 Требования

Студенты должны владеть следующими знаниями, умениями и навыками для выполнения задания.

- Разработка модульных приложений на языке C++.
- Понимание концепций *интерфейса* и *реализации* и механизмов поддержки этих концепций в языке C++.
- Владеть основами ООП: описание и определение классов, описание интерфейсов в виде чисто абстрактных классов.
- Знать основы стандартной библиотеки C/C++.
- Владеть основами ввода/вывода с реализацией на C/C++.
- Знать абстрактный тип данных (АТД) «стек», его основные операции.

1.2 Цели и задачи

Основная цель работы — изучить вопросы имплементации АТД «стек» на задачах, где этот АТД традиционно используется в качестве основной структуры данных.

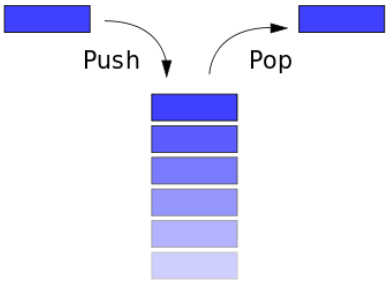

Рис. 1: Стек.

Задачи для выполнения в рамках самостоятельной работы:

- Изучить способы имплементации АТД «стек» (рис. 1).
- Изучить концепцию стекового языка по предложенным ссылкам.
- Изучить концепцию стековой машины, разбирающий выражения в виде стекового языка.
- Изучить *прямую* и *обратную польские записи* арифметических выражений.
- Определить класс, реализующий стек элементов — целых чисел.
- Определить классы и интерфейсы, реализующие стековую машину и объекты, реализующие операции на этой машине, с использованием технологии динамического расширения множества операций.
- Разработать пользовательское приложение, запрашивающее у пользователя ввод из стандартного потока ввода выражения в виде строки текста и выполняющее разбор и оценку этого выражения с исполь-

зованием разработанной стековой машины с последующим выводом результата в стандартный поток вывода.

1.3 Ожидаемые результаты

Студенты, успешно[1] выполнившие задание, получат навыки разработки динамически расширяемых компонент, стековых структур, машин для разборки простейших выражений и общие компетенции по разработке приложений на языке C++.

[1] И самостоятельно.

2 Предпосылки

Стек (*stack*, *push-down stack*) представляет упорядоченную коллекцию элементов, добавление новых элементов в которую и извлечение ранее добавленных осуществляется в самый конец коллекции, называемый *вершиной стека* [Ахо et al.(2000)Ахо, Хопкрофт, Ульман].

Стек находит многочисленные применение в computer science, особенно — в трансляторах, виртуальных машинах и реальных исполнителях[2]. Так, для организации вызова подпрограмм ассемблера процессоров Intel стек используется для передачи аргументов в вызываемую подпрограмму, получения результатов ее выполнения и сохранения контекста процесса в точки вызова подпрограммы. На этом в частности основан принцип рекурсии, при котором при каждом рекурсивном вызове методом самого себя его текущее состояние сокраняется в стеке, а при возврате из метода — выталкивается из стека.

[2] К ним в частности относятся микропроцессоры, построенные по архитектуре IA-32, IA-64 и др.

Стековым языком программирования является такой язык, в котором для передачи параметров используется машинная модель стека. Этому описанию соответствует несколько языков, в первую очередь Forth и PostScript, а также многие ассемблерные языки (использующие эту модель на низком уровне — Java, C#).

Выполнение программы в стековом языке программирования представляет собой операции на одном или нескольких стеках, которые могут иметь различное предназначение. Вследствие этого программные конструкции других языков программирования должны быть изменены, прежде чем они могут быть использованы в стековом языке[3].

[3] `https://ru.wikipedia.org/wiki/Стековый_язык`

Стековые языки программирования используют так называемую «обратную польскую» нотацию (англ. RPN, reverse polish notation), или *постфиксную нотацию*.

2.1 Обратная польская нотация

Обратная польская нотация (ОПН) — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций [Ахо et al.(2003)Ахо, Сети, Ульман].

Отличительной особенностью обратной польской нотации является то, что все аргументы (или операнды) расположены перед знаком операции. В общем виде запись выглядит следующим образом:

- Запись набора операций состоит из последовательности операндов и знаков операций. Операнды в выражении при письменной записи разделяются пробелами.
- Выражение читается слева направо. Когда в выражении встречается знак операции, выполняется соответствующая операция над двумя последними встретившимися перед ним операндами в порядке их записи. Результат операции заменяет в выражении последовательность её операндов и её знак, после чего выражение вычисляется дальше по тому же правилу.
- Результатом вычисления выражения становится результат последней вычисленной операции.

В качестве примера рассмотрим вычисление выражения `7 2 3 * –` (эквивалентное выражение в инфиксной нотации: 7–2*3).

1. Первый по порядку знак операции — «*», поэтому первой выполняется операция умножения над операндами 2 и 3 (они стоят последними перед знаком). Выражение при этом преобразуется к виду `7 6 –` (результат умножения — 6 — заменяет тройку «2 3 *»).
2. Второй знак операции — «–». Выполняется операция вычитания над операндами 7 и 6.
3. Вычисление закончено. Результат последней операции равен 1, это и есть результат вычисления выражения.

Преимуществом ОПН перед прямой польской нотацией[4] заключается в естетсвенном расширении обратной польской записи на унарные, тернарные и операции с любым другим количеством операндов: при использовании знаков таких операций в вычислении выражения операция применяется к соответствующему числу последних встретившихся операндов.

[4] *Прямая польская нотация* — запись выражения, в котором операторы стоят перед операндами.

Важные особенности обратной польской нотации:

- Порядок выполнения операций однозначно задаётся порядком следования знаков операций (операторов) в выражении, поэтому нет необходимость в использовании группирующих скобок[5] и введения приоритетов и ассоциативности операций.
- В отличие от инфиксной записи[6], невозможно использовать одни и те же знаки для записи унарных и бинарных операций. Так, в инфиксной записи выражение `5 * (–3 + 8)` использует знак «минус» как символ унарной операции (изменение знака числа), а выражение `(10 - 15) * 3` применяет этот же знак для обозначения бинарной операции (вычитание). Конкретная операция определяется тем, в какой позиции находится знак. Обратная польская запись не позволяет этого: запись `5 3 – 8 + *` (условный аналог первого выражения) будет интерпретирована как ошибочная, поскольку невозможно определить, что «минус» после

[5] Из-за отсутствия скобок обратная польская запись короче инфиксной.

[6] «Традиционная» запись арифметического выражения в виде `(1 + 2) * 3`.

5 и 3 обозначает не вычитание; в результате будет сделана попытка вычислить сначала `5 - 3`, затем `2 + 8`, после чего выяснится, что для операции умножения не хватает операндов. Чтобы всё же записать это выражение, придётся либо переформулировать его, либо ввести для операции изменения знака отдельное обозначение, например, «±»: `5 3 ± 8 + *`.

- Так же, как и в инфиксной нотации, в ОПН одно и то же вычисление может быть записано в нескольких разных вариантах. Например, выражение `(10 - 15) * 3` в ОПН можно записать как `10 15 - 3 *`, а можно — как `3 10 15 - *` .

2.2 Стековая машина

Стековой машиной называется алгоритм, проводящий вычисления по обратной польской записи [Ахо et al.(2003)Ахо, Сети, Ульман].

Автоматизация вычисления выражений в обратной польской нотации основана на использовании стека. Алгоритм вычисления для стековой машины описывается следующим образом:

1. Обработка входного токена[7]:
   - Если на вход подан операнд, он помещается на вершину стека.
   - Если на вход подан знак операции, то соответствующая операция выполняется над требуемым количеством значений, извлечённых из стека, взятых в порядке добавления. Результат выполненной операции кладётся на вершину стека.
2. Если входной набор токенов обработан не полностью, перейти к шагу 1.
3. После полной обработки входного набора символов результат вычисления выражения лежит на вершине стека.

[7] *Токеном* будем называть последовательность символов (char), не разделенных пробелами и содержащими цифры, точки и знаки операций, введенных в стековую машину.

3 Описание задания

В рамках самостоятельной работы студентам предлагается разработать расширяемую по множеству операций стековую машину, разбирающую и оценивающую выражение, представленное в ОПН. Машина реализуется в виде программы на языке C++.

Арность операций — унарные, бинарные или тернарные. В качестве символа оператора допустимо использовать любой **однобайтный** символ, не используемый для представления чисел[8] в ЭВМ. Операнды и результат операций — целые числа.

Машина получает на вход строку символов, производит разделение ее на токены, последовательно анализирует токены, разделяя их на числа-операнды и операторы; операнды помещаются на стек, для операторов ищется сопоставление в множестве зарегистрированных операторов и, если таковое сопоставление находится, из стека извлекается необходимое

[8] Строго говоря, использование символов `+` и `-` следовало бы считать неразрешенным в рамках данного ограничения, однако, пользуясь особенностью, что запись токена, состоящего из одного такого символа, не может считаться полноценным числом — и функции преобразовании строки в число не посчитают этот символ числом — это ограничение фактически не распространяется на любые символы, кроме цифр.

число операндов (с учетом арности операции), вызывается соответствующая операция и результат помещается на стек.

В программе предусматривается возможность не выполнения (или выполнение) очищения стека между вызовами, что может использоваться для обмен результатами вычислений в стеке между двумя вызовами цикла работы машины.

3.1 Разрабатываемые типы и подпрограммы

Класс `IntStack`. Реализует целочисленный стек, используемой стековой машиной для своей работы.

Класс содержит основные операции, определяемые для АТД «стек»[9]. Размер стека определяется опциональным параметром конструктора, значение которого равно по умолчанию встроенной константе `STACK_SIZE`. Внутреннее представление стека реализовано с помощью C-style целочисленного массива целых чисел, создаваемого динамически в конструкторе класса стека с последующем освобождением в деструкторе этого класса.

[9] Для абстрагирования определения пустого и заполненного стеков от реального представления элементов в памяти, класс снабжен двумя «неклассическими» методами `isEmpty()` и `isFull()` соответственно.

Интерфейсный класс `IOperation`. Описывает интерфейс объекта, реализующего операцию.

Тип описывает два чисто виртуальных метода, необходимых для реализации любым объектом, способным выступать в роли операции.

- `Arity` — тип-перечисление, описывающие возможные операции: унарные, бинарные или тернарные.
- `operation()` — метод, получающий на вход символ операции[10] и от одного до трех целочисленных параметров, выполняющих непосредственно операцию и возвращаюий результат операции по значению.
- `getArity()` — метод, возвращающий арность конкретной операции.

Класс описывает свой деструктор в виде непубличного члена, таким образом запрещая управлять временем жизни полиморфных объектов через указатель на этот (базовый) класс[11].

Примером класса, реализующего интерфейс `IOperation`, является класс `PlusOp`. Он реализует бинарную операцию «сложение», сопоставленную с оператором «плюс» (+) в обычном арифметическом смысле.

[10] *Design Rationale*: вопрос «для чего передавать символ операции, если отображение произведено на этапе добавления операции?» закономерен; чтобы ответить на него, нужно посмотреть, как можно зарегистрировать несколько операций. Если мы не передаем символ операции параметром, то понадобилось бы определять столько типов/объектов, сколько операций мы хотим передавать. В приведенном же варианте дизайна мы можем одним типом реализовать сразу несколько операций, что удобно.

[11] Подробнее см. Herb Sutter: `http://www.gotw.ca/publications/mill18.htm`

Класс `StackMachine`. Реализует саму стековую машину.

Стековая машина содержит в виде поля класса рабочий стек, который может сохранять значения между вызовами процедуры исполнения машины. Класс машины содержит следующие основные элементы:

- Тип `SymbolToOperMap` — псевдоним для типа-отображения символа операции на объект, реализующий эту операцию. Связанный тип `SymbolToOperMapConstIter` — псевдоним для типа, реализующей концепт [Austern(1999)] итератора стандартной библиотеки для типа `SymbolToOperMap`;

- `registerOperation()` — метод регистрации объекта-операции, сопоставленной некоторому знаку оператора;
- `getOperation()` — метод получение объекта-операции по параметру — знаку оператора;
- `calculate()` — метод, вычисляющий переданное выражение в форме ОПН;
- `getStack()` — методы, возвращающие стек вычислений;
- `_opers` — поле, представляющее коллекцию зарегистрированных операций;
- `_s` — поле, представляющее стек операций.

3.2 Замечания по реализации

Для регистрации объекта-операции используется метод `registerOperation()`, реализация которого представлена в листинге 1.

Листинг 1. Реализация метода `StackMachine::registerOperation()`.

```
1  void StackMachine::registerOperation(char symb, IOperation* oper)
2  {
3      SymbolToOperMapConstIter it = _opers.find(symb);
4      if(it != _opers.end())
5          throw std::logic_error("An operation ... is alr. reg...");
6
7      _opers[symb] = oper;
8  }
```

Метод `StackMachine::registerOperation()` демонстрирует использование метода `find()` типа данных `std::map`[12] для отыскания значения по ключу. Если значение не присутствует в отображении, возвращаемое значение метода — итератора — будет равен `_opers.end()`[13]. В приведенном методе добавить повторно для одного и того же символа-оператора другой объект-операцию нельзя, и такая ситуация приводит к генерированию исключительной ситуации.

[12] Близким аналогичным типом данных языка C# является `Dictionary`.

[13] Стандартно end() представляет т.н. «элемент после последнего», то есть не существующий квазиэлемент, указывающий на то, что некоторая коллекция перебрана полностью или пуста.

Пример подключения операции «сложения» к объекту стековой машины представлен в листинге 2. Необходимо обратить внимание, что при регистрации передается адрес объекта, так как метод регистрации ожидает указатель на `IOperation` или любого производного класса.

Листинг 2. Подключение объекта операции «сложение» к стековой машине.

```
1      StackMachine sm;                    //  Создание объекта стековой машине (да, тоже в стеке :)
2
3      PlusOp plusop;                      //  Создание объекта, реализующего операцию сложение
4
5      sm.registerOperation('+', &plusop); //  Регистрация объекта-сложения.
```

Метод `StackMachine::getOperation()` для получения по заданному символу объекта-операции студентам предлагается разработать самостоятельно, используя аналогичным образом операцию `find()`[14].

[14] Важное замечание: разыменования итератора структуры `map` возвращает пару `std::pair`, состоящую из *ключа* и *значения*. Ключом в данном случае будет выступать объект типа *char* — символ операции, а значением — указатель на объект операции типа `IOperation*`. Для извлечения из итератора `it` второй компоненты — *значения* — можно использовать такое выражение: `it->second`.

Разбор выражения. При выполнении разбора выражения рекомендуется предварительно выполнить разбиение строки на составляющие токены, используя в качестве символа-разделителя пробел. Полученные токены поместить в любую структуру, например `std::list`.

Другим вариантом является проход по строке выражения с выделением подстроки «от пробела до пробела». Для получения строки в стиле C[15] из строки `std::string s` можно воспользовать следующим методом: `s.c_str()`.

При вызове операции в зависимости от арности со стека нужно вытолкнуть 1, 2 или 3 верхних элемента, соответствующим операндам. Значения следует передать в качестве параметров при вызове метода `IOperation::operation()`. При этом 2-й и 3-й операнды этого метода являются опциональными и по умолчанию равны нулю. Подразумевается, что операции низкой арности просто не используют эти параметры.

[15] Т.н. null-terminated строки, то есть последовательности символов, заканчивающиеся «признаком конца» — символом `'\0'`.

Преобразования строки в число. Рекомендуется разработать функцию (в виде свободной функции или статического метода класса `StackMachine`), получающего на вход строку и возвращающего истину, если удалось преобразовать ее в целое число (а само число — через параметр по ссылке, например, или в виде пары `std::pair` вместе с булевым флагом), и ложь — если это не число. Для преобразования можно использовать C-style функцию `strtol()`, либо объекты строковых потоков стандартной библиотеки[16].

[16] Удобные обобщенные методы `lexical_cast` предоставляет библиотека boost, однако ее использование в данной работе не разрешается.

3.3 Пользовательские операции

В зависимости от варианта, назначаемого преподавателем индивидуально, каждому студенту необходимо в добавок к предложенной операции «сложение» реализовать операции из следующего набора. Арность операции определяется аргументами x, y и z. Расширяемые операции должны быть реализованы путем разработки новых классов, реализующих интерфейс `IOperation`[17].

1. Операция вычитания `-` : $x - y$. Результат — арифметическая разность двух операндов.
2. Операция умножения `*` : $x * y$. Результат — арифметическое произведение двух операндов.
3. Операция деления `/` : x/y. Результат — арифметическое частное двух операндов при ненулевом втором. Если второй операнд равен нулю, результат определяется константой `INT_MAX` со знаком первого операнда.
4. Операция выбора операнда `?` : $x?y : z$. Результат — операнд 2, если операнд 1 ненулевой, иначе операнд 3.
5. Операция повторения операнда `=` : $= x$. Результат — значение операнда 1.

[17] Требование подразумевает регистрацию операторов по аналогии с предложенным классом `PlusOp`. Студент имеет возможность по выбору разработать по одному классу-расширению на операцию, либо все операции, кроме +, реализовать в едином классе-расширении. Тестирование стековой машины будет осуществляться в режиме «черного ящика», поэтому именование и внутренняя структура и именование классов-расширений значения не имеют.

6. Операция изменения знака операнда `!` : $!x$. Результат — значение операнда 1 с противоположным знаком.
7. Операция инверсии битов операнда `~` : $\sim x$. Результат — значение операнда 1 в обратном коде.
8. Операция логического побитового умножения (AND) операндов `&` : $x \& y$.
9. Операция логического побитового сложения (OR) операндов `|` : $x|y$.
10. Операция возведения в степень числа, мантисса которого определяется первым оператором, показатель — вторым `^` : x^y.

4 Указания по выполнению задания

4.1 Исходные файлы

Студенты получают для работы следующие файлы исходных кодов на языке C++.

- `int_stack.h` — описание интерфейса класса целочисленного стека.
- `stack_machine.h` — описание интерфейсов классов `IOperation`, `PlusOp` и `StackMachine`.
- `stack_machine.cpp` — реализация некоторых методов классов `PlusOp` и `StackMachine`.

4.2 Задачи для выполнения

Студенты должны реализовать следующие компоненты программы.

- `int_stack.cpp` — имплементационную часть интерфейса `IntStack.h` целочисленного стека.
- `stack_machine.cpp` — реализацию всех нереализованных методов соответствующих типов, описанных в интерфейсной части модуля.
- `main.cpp` — точка входа в приложения и два тестовых метода (для стека и стековой машины соответственно). Реализация задачи должна проходить эти тесты как минимум.

Кроме реализации предложенных интерфейсов студенты реализуют необходимые классы, имплементирующие операторы из списка 3.3[18].

4.3 Представление результатов

Студенты представляют разработанные файлы[19] (а также измененные файлы, если это допускается заданием) модулей в виде зафиксированного состояния git-репозитория рабочего проекта (commit-a), отправленного на сервер для автоматизированной проверки (git push).

[18] Полный перечень реализуемых операций назначается индивидуально каждым преподавателем персонально каждому студенту или подгруппе целиком.

[19] Важно не забывать своевременно добавлять новые файлы в git-индекс, в противном случае они не будут участвовать в версионировании и в дальнейшем не поступят на сервер, что приведет к невозможности проверки.

Список литературы

[Ахо et al.(2003)Ахо, Сети, Ульман] А. В. Ахо, Р. Сети, Д. Д. Ульман. *Компиляторы: принципы, технологии и инструменты.* М.: «Вильямс», 2003.

[Ахо et al.(2000)Ахо, Хопкрофт, Ульман] А. Ахо, Дж. Хопкрофт, Дж. Ульман. *Структуры данных и алгоритмы. Пер. с англ. Уч. пос.* Издательский дом «Вильямс», 2000.

[Austern(1999)] M. H. Austern. *Generic Programming and the STL*. Professional computing series. Addison-Wesley, 1999.