**Дисциплина «Информатика»**

**Рабочая тетрадь № 8**

|  |
| --- |
| Основной задачей компьютера является обработка данных. Под обработкой данных понимается как вычислительные процедуры, так и процедуры, преобразующие одни данные в другие данные.  Данные – это форма представление информации, доступная для обработки. Формат представления данных – это последовательность бит (байт). Физически данные хранятся в виде файлов или потоков данных на физических носителях информации. Поток данных – это абстракция для доступа к данным из файлов, периферийных устройств и т.д.  Список – структура данных, состоящая из узлов, хранящих как данные, так и ссылку на следующий элемент. |

|  |
| --- |
| **1.Теоретический материал** |
| При программировании серьезных проектов необходимо работать не только с переменными, но и со составными структурами данных. Простейшей структурой данных является массив.  Массив – структура данных, хранящая набор однотипных переменных, доступных по индексу.    Рисунок 1 – Схема массива  Список – структура данных, состоящая из узлов, хранящих как данные, так и ссылку на следующий элемент.    Рисунок 2 – Схема динамического списка  Часто нам нужно организовать очередь, когда мы сохраняем и имеем доступ к элементам не по индексу, а по принципу «первый вошел, первый вышел» FIFO – «*firstin, firstout*».    Рисунок 3 – Схема очереди  Еще одной важной структурой данных является стек.Стек представляет собой структуру данных, в которой принцип: «последний вошел, первый вышел» LIFO – «*lastin, firstout*».    Рисунок 4 – Схема стека |

|  |  |
| --- | --- |
| **2. Пример** | |
| ***Задача:*** | |
|  | Используя стек, напечатать символы некоторой величины строкового типа в обратном порядке. |
| ***Решение:*** | |
|  | С использованием стека задача решается следующим образом. Каждый символ заданной величины, начиная с первого, размещаем в стеке. В результате на вершине стека окажется последний символ, “под” ним — предпоследний и т.д. Извлекая символы из стека, выводим их на экран в порядке, обратном исходному.  В программе на школьном языке программирования (“КуМир”) используем два вспомогательных алгоритма для работы со стеком:  — Поместить — реализует размещение в стеке нового символа;  — Извлечь — осуществляет извлечение элемента из стека (извлекаемое значение является результатом алгоритма-функции символьного типа).  Учтем, что программная реализация этих алгоритмов значительно упрощается, если добавление нового значения в стек и извлечение значения на его вершине проводить в конце заполненной части массива:  **алг** Поместить(**арг сим** симв)  **нач**  **·**|*Увеличиваем значение величины вершина*  **·**вершина := вершина + 1  **·**|*Записываем на вершину стека*  **·**|*очередной* *символ*  **·**стек[вершина] := симв  **кон**  **алг** **сим** Извлечь  **нач**  **·**|*Значение функции*  **·**знач: = стек[вершина]  **·**|*Уменьшаем значение величины вершина*  **·**вершина := вершина — 1  **кон**  где:  — *стек* — массив, моделирующий стек; поскольку в нем хранятся отдельные символы, то описание этого массива должно быть таким:  **цел** nmax; nmax := 100  **сим** **таб** стек[1:nmax]  *— вершина —*индекс элемента массива, являющегося вершиной стека; описание этой величины:  **цел** вершина; вершина := 0  Создадим также функцию, возвращающую результат логического типа в зависимости от того, пуст стек (*истина*) или нет (*ложь*):  **алг лог** СтекПуст  **нач**  **· знач** := вершина = 0  **кон**  С использованием созданных вспомогательных алгоритмов главная программа принимает вид:  **алг** Строка в обратном порядке  **начлит** строка, **цел** i  **· вывод** "Введите строку символов "  **· ввод** строка  **·**|*Размещаем символы в стеке*  **· нц для**i **от** 1 **до** длин(строка)  **· ·**Поместить(строка[i])  **· кц**  **· вывод** **нс**, "Строка в обратном  порядке символов:"  **·**|*Извлекаем символы из стека*  **· нц пока не** СтекПуст  **· · вывод** Извлечь  **· кц**  **кон** |
| ***Задача\*:*** | |
|  | Написать программу, которая определяет, является ли введенная скобочная структура правильной. Примеры правильных скобочных выражений: (), (())(), ()(), ((())), неправильных — )(, ())((), (, )))), ((()). |
| ***Решение:*** | |
|  | Можно рассуждать так. Рассматриваем последовательно каждый символ заданной величины слева направо. Если очередной символ — левая скобка, то размещаем ее в стеке, если правая — то извлекаем элемент из стека (это обязательно должна быть левая скобка). После рассмотрения всей строки, если все правильно, стек должен оказаться пустым.  Обсудим особенности вспомогательных алгоритмов.  Так как в стеке размещается только левая скобка, то алгоритм *Поместить* можно оформить как алгоритм без параметров:  **алг** Поместить  **нач**  **·**вершина := вершина + 1  **·**стек[вершина] := "("  **Кон**  Поскольку в отличие от первой задачи здесь значение на вершине не используется, то алгоритм *Извлечь* целесообразно оформить не в виде функции:  **алг** Извлечь  **нач**  **·**|*Уменьшаем значение величины вершина*  **·**вершина := вершина — 1  **кон**  Однако при таком оформлении алгоритма *Извлечь*главная программа:  **цел** nmax; nmax := 30  **сим таб** стек[1:nmax]  **цел** вершина  вершина := 0  **алг** Проверка правильности записи скобок  **начлит** выражение, **цел** i  **· вывод** "Введите скобочное выражение "  **· ввод** выражение  **· нц для** i **от** 1 **до** длин(выражение)  **· · если** выражение[i] = "("  **· · · то**  **· · · ·**Поместить  **· · все**  **· · если** выражение[i] = ")"  **· · · то**  **· · · ·**Извлечь  **· · все**  **· кц**  **· если** СтекПуст  **· · то**  **· · · вывод нс**, "Выражение правильное"  **· · иначе**  **· · · вывод нс**, "Выражение неправильное"  **· все**  **Кон**  — будет работать неправильно (убедитесь в этом, проанализировав приведенные далее разъяснения!). Дело в том, что когда в выражении встретится “неправильная” правая скобка, то при ее извлечении из стека значение величины *вершина* станет отрицательным, и последующее размещение в стеке левой скобки провести не удастся (при выполнении программы появится сообщение об ошибке, связанное с выходом за левую границу массива). Ввести в алгоритм *Извлечь* вывод на экран сообщения “Стек пуст!”, как это делалось в статье [1], нельзя — при этом ошибка сохранится.  Чтобы зафиксировать факт появления “неправильной” правой скобки, введем в заголовок процедуры *Извлечь* величину логического типа с именем *успешно* (имеется в виду, что извлечение элемента на вершине прошло успешно). Пока ошибка не найдена, то *успешно* *= истина*(на школьном языке программирования значение *истина*обозначается как *да*), если найдена — *успешно* *= ложь*(*нет*).  **алг** Извлечь(**арг рез лог**успешно)[2](https://inf.1sept.ru/articles/2009/23/10#2)  **нач**  **· если** вершина > 0  **· · то**  **· · ·**|*"Извлекаем"*  **· · ·**вершина := вершина — 1  **· · ·**|*Извлечение прошло успешно*  **·** **·** **·** успешно := **да**  **· иначе**  **· ·**|*Извлечь нельзя*  **·** **·** успешно := **нет**  **· все**  **Кон**  C учетом этого в главной программе следует использовать оператор цикла с предусловием:  **алг** Проверка правильности записи скобок  **нач лит** выражение, **цел** i, **лог** успешно  **· вывод** "Введите скобочное выражение "  **· ввод** выражение  **·** успешно := **да** |*Условно*  **·**i := 1  **· нц пока**i <= длин(выражение) **и** успешно  **· · если** выражение[i] = "("  **· · · то**  **· · · ·**Поместить  **· · все**  **· · если** выражение[i] = ")"  **· · · то**  **· · · ·**Извлечь(успешно)  **· · все**  **· ·**i := i + 1  **· кц**  **· если** СтекПуст  **· · то**  **· · · вывод нс**, "Выражение правильное"  **· · иначе**  **· · · вывод нс**, "Выражение неправильное"  **· все**  **кон** |

|  |  |  |
| --- | --- | --- |
| **3. Задания** | | |
| 1. | ***Задача:*** | |
|  | Дана величина*a*строкового типа из четного количества символов. Получить и напечатать величину*b*, состоящую из символов первой половины величины*a*, записанных в обратном порядке, после которых идут символы второй половины величины*a*, также записанные в обратном порядке. Например, при*а*= “привет”*b*должно быть равно “ипртев”. |
| ***Решение:*** | |
|  | **алг** Поместить(**арг сим** симв)  **нач**  вершина := вершина + 1  стек[вершина] := симв  **кон**  **алг** **сим** Извлечь  **нач**  знач: = стек[вершина]  вершина := вершина — 1  **кон**  **цел** nmax; nmax := 100  **сим** **таб** стек[1:nmax]  **цел** вершина; вершина := 0  **алг лог** СтекПуст  **нач**  **знач** := вершина = 0  **кон**  **алг** ЧастичныйРеверс  **нач лит** слово, **цел** i  **ввод** слово  **нц для**i **от** 1 **до** (длин(строка)/ 2)  Поместить(слово[i])  **кц**  **нц пока не** СтекПуст  **вывод** Извлечь  **кц**  **нц для**i **от** (длин(строка)/ 2) **до** длин(строка)  Поместить(слово[i])  **кц**  **нц пока не** СтекПуст  **вывод** Извлечь  **кц**  **кон** |
| 2. | ***Задача\*:*** | |
|  | Проверить правильность расстановок скобок в выражении, использующем скобки вида: “(”, “)”, “{”, “}”, “[”, “]”. Требование вложенности скобок разного вида не учитывать, т.е., например, последовательности скобок “({})[]” и “([{()}]}” — правильные. |
| ***Решение:*** | |
|  | **алг** Поместить  **нач**  **·**вершина := вершина + 1  **·**стек[вершина] := "("  **Кон**  **алг** Извлечь(**арг рез лог**успешно)  **нач**  **· если** вершина > 0  **· · то**  **· · ·**вершина := вершина — 1  **·** **·** **·** успешно := **да**  **· иначе**  **·** **·** успешно := **нет**  **· все**  **Кон**  **алг** Проверка правильности записи скобок  **нач лит** выражение, **цел** i, **лог** успешно  **· вывод** "Введите скобочное выражение "  **· ввод** выражение  **·** успешно := **да**  **·**i := 1  **· нц пока**i <= длин(выражение) **и** успешно  **· · если** выражение[i] = "("  **· · · то**  **· · · ·**Поместить  **· · все**  **· · если** выражение[i] = ")"  **· · · то**  **· · · ·**Извлечь(успешно)  **· · все**  **· ·**i := i + 1  **· кц**  **·** успешно := **да**  **·**i := 1  **· нц пока**i <= длин(выражение) **и** успешно  **· · если** выражение[i] = "{"  **· · · то**  **· · · ·**Поместить  **· · все**  **· · если** выражение[i] = "}"  **· · · то**  **· · · ·**Извлечь(успешно)  **· · все**  **· ·**i := i + 1  **· кц**  **·** успешно := **да**  **·**i := 1  **· нц пока**i <= длин(выражение) **и** успешно  **· · если** выражение[i] = "["  **· · · то**  **· · · ·**Поместить  **· · все**  **· · если** выражение[i] = "]"  **· · · то**  **· · · ·**Извлечь(успешно)  **· · все**  **· ·**i := i + 1  **· кц**  **· если** СтекПуст  **· · то**  **· · · вывод нс**, "Выражение правильное"  **· · иначе**  **· · · вывод нс**, "Выражение неправильное"  **· все**  **кон** |

|  |  |  |
| --- | --- | --- |
| **Тест** | | |
| **1.** | Какой этап НЕ входит в этапы создания программы: | |
|  | А) проектирование Б) бенчмаркинг  В) написание кода Г) тестирование и отладка |
| ***Ответ:*** | |
|  | Б |
| **2.** | Какиепарадигмы программирования существуют (множественный выбор) | |
|  | А) структурное программирование  Б) объектно-ориентированное  В) функциональное программирование  Г) все ответы верны |
| ***Ответ:*** | |
|  | Г |
| **3.** | Что НЕ относится к принципам структурного программирования | |
|  | А) Программа представляется в виде иерархической структуры блоков  Б) Разработка программы ведется методом «сверху вниз»  В) Разработка программы ведется методом «снизу вверх»  Г) Тело программы содержит управляющие структуры: последовательность, ветвление, цикл и подпрограммы |
| ***Ответ:*** | |
|  | В |
| **4.** | Линейный список, в котором доступен только последний элемент, называется | |
|  | А) стеком Б) очередью  В) двухсторонней очередью Г) кольцом |
| ***Ответ:*** | |
|  | А |
| **5.** | Структура данных,работа с элементами которой организована по принципу FIFO (первый пришел - первый ушел) это – | |
|  | А) Стек Б) Двухсторонняя очередь  В) Очередь Г) Список |
| ***Ответ:*** | |
|  | В |
| **6.** | В чём сосбенности стека ? | |
|  | А) открыт с обеих сторон на вставку и удаление  Б) доступен любой элемент  В) открыт с одной стороны на вставку и удаление  Г) закрыт с обеих сторон на вставку и удаление |
| ***Ответ:*** | |
|  | В |
| **7.** | Дана следующая последовательность стековых операций. Что будет на вершине стека, когда последовательность завершится?  m = Stack()  m.push('x')  m.push('y')  m.pop()  m.push('z')  m.peek() | |
|  | А) 'x' Б) 'y' В) 'z' Г) стек пуст |
| ***Ответ:*** | |
|  | В |
| **8.** | Из всех приведенных ниже утверждений, касающихся объектов и классов, отметьте правильные. | |
|  | 1) значения полей описывают состояние объекта  2) методы класса описывают поведение объектов  3) поля класса могут быть разного типа  4) поля класса должны быть разного типа  5) в любом классе должно быть как минимум одно поле данных |
| ***Ответ:*** | |
|  | 1,2,3 |
| **9.** | 1. Термин "наследование" обозначает, что... | |
|  | А) В производных классах присутствует часть состояния родительского класса  Б) Производные классы содержат поля и методы родительского  В) Производные классы наследуют модификаторы доступа членов родительского класса |
| ***Ответ:*** | |
|  | Б |
| **10.** | 1. Инкапсуляция – это : | |
|  | А) Свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью.  Б) Сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса или копирования прототипа (например, после запуска результатов компиляции и связывания исходного кода на выполнение)  В) Свойство системы, позволяющее объединить данные и методы, работающие с ними в классе, и скрыть детали реализации от пользователя. |
| ***Ответ:*** | |
|  | В |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **Б** | **Г** | **В** | **А** | **В** | **В** | **В** | **1,2,3** | **Б** | **В** |

**Реализация задач на языке программирования Python**

|  |
| --- |
| В предыдущих разделах мы рассматривали традиционное программирование на Python, когда вся программа разбивается (или не разбивается) на отдельные модули, содержащие функции. Такое программирование соответствует парадигме структурного программирования. Само структурное программирование оказалось колоссальным шагом в построении программ. Однако еще большим шагом является парадигма объектно-ориентированного программирования. В этом подходе программа состоит из отдельных классов, которые объединяют в себе как переменные, называемые полями класса, так и функции, называемые методами класса. |

|  |
| --- |
| **1. Теоретический материал** |
| Объектно-ориентированное программирование состоит из трех китов:   * инкапсуляция * наследование * полиморфизм   Рассмотрим на примерах эти понятия. Первое - инкапсуляция - это объединение в одном объекте данных и программного кода таким образом, что для внешней работы внутренняя часть объекта может быть скрыта от пользователя. Инкапсуляция может быть реализована не только с помощью классов, но и с помощью модулей, но классы позволяют сделать инкапсуляцию естественным путем.  Создадим класс в Python. Для этого необходимо определить класс (новый тип данных) и создать объект, называемый экземпляром класса. Мы рекомендуем имена классов начинать с заглавной буквы "T", подчеркивая тем самым, что речь идет о типе данных. Делаетсяэтотак:  class TAnimal:  name = ""  def \_\_init\_\_(self, name):  self.name = name  def say(self):  print(self.name)  Теперь создадим экземпляр этого класса. Экземпляр класса представляет собой переменную, с которой можно работать обычным образом.  Animal = TAnimal("Обезьяна")  Animal.say()  Рассмотрим синтаксис Python при создании классов. Все начинается с ключевого слова**class**. далее в блоке из отступов мы определяем переменные, которые будем называть полями и функции, которые называются методами. Методы определяются, как обычные функции и могут возвращать значения. Единственное отличие состоит в том, что у всех методов есть обязательный первый параметр, который по традиции всегда называем**self**в котором передается ссылка на экземпляр класса. Поэтому когда внутри класса метод хочет обратиться к своему полю, то необходимо использовать конструкцию**self.name**. Заметим, что при вызове методов мы первый параметр не задаем.  Далее, у каждого класса есть метод, с именем**\_\_init\_\_**, который называется конструктором класса. Этот метод вызывается в момент создания экземпляра**Animal = TAnimal("Обезьяна")**. Конструктор может иметь любое количество параметров.  Предположим, что теперь нам нужно сделать класс для описания конкретного животного - кошки. Дляэто мы используем наследование классов, когда можно определять новые классы, как наследники существующих. При этом новый класс будет иметь все поля и методы наследуемого класса. Воткакэтоделается:  class TAnimal:  name = ""  def \_\_init\_\_(self, name):  self.name = name  def say(self):  print(self.name)    class TCat(TAnimal):  def may(self):  print("Мяу!")    Cat = TCat("Кошка")  Cat.say()  Cat.may()  Мы видим, что у наследованного класса сохранился конструктор и метод**say**.  В последнем примере мы выдели, что наследный класс, также как и исходный имеет конструктор, который принимает в качестве параметра - название животного тогда, что в данном случае излишне. Для решения этой проблемы мы воспользуемся объектно-ориентированным механизмом - полиморфизмом. Полиморфизм - это возможность замены методов при наследовании. Сделаем так, чтобы не нужно было передавать в конструкторе название "Кошка".  class TCat(TAnimal):  def \_\_init\_\_(self):  super().\_\_init\_\_("Кошка")  def may(self):  print("Мяу!")    Cat = TCat()  Cat.say()  Cat.may()  Результат выполнения этой программы будет аналогичный, но теперь при использовании этого класса нам не нужно передавать в конструкторе никаких параметров. Полиморфное перекрытие методов делается простым объявлением метода (в данном случае конструктора). При этом не можно менять входные параметры. Если в результате написания кода метода возникает необходимость вызвать перекрытый метод, то для этого необходимо использовать функцию**super()**, которая по сути просто возвращает ссылку на родительский класс. Самое удивительное в полиморфизме, что изменяя метод, он меняется даже когда на него есть ссылки родительского класса. Рассмотрим еще один пример. Пусть у нас есть класс:  classTDo:  def Operation(self, x, y):  return x + y  def Run(self):  x = int(input("Enter x > "))  y = int(input("Enter y > "))  z = self.Operation(x, y)  print("Result = " + z.\_\_str\_\_())    Do = TDo()  Do.Run()  С помощью полиморфизма заменим функцию**Operation**на другую в наследном классе:  class TDo2(TDo):  def Operation(self, x, y):  return x \* y |

|  |  |
| --- | --- |
| **2. Пример** | |
| ***Задача:*** | |
|  | Необходимо разработать виртуальную модель процесса обучения. В программе должны быть объекты-ученики, учитель, кладезь знаний. |
| ***Решение (кодпрограммы):*** | |
|  | Потребуется три класса – "учитель", "ученик", "данные". Учитель и ученик во многом похожи, оба – люди. Значит, их классы могут принадлежать одному надклассу "человек". Однако в контексте данной задачи у учителя и ученика вряд ли найдутся общие атрибуты.  Определим, что должны уметь объекты для решения задачи "увеличить знания":   * Ученик должен уметь брать информацию и превращать ее в свои знания. * Учитель должен уметь учить группу учеников. * Данные могут представлять собой список знаний. Элементы будут извлекаться по индексу.   class Data:  def \_\_init\_\_(self, \*info):  self.info = list(info)  def \_\_getitem\_\_(self, i):  return self.info[i]    class Teacher:  def teach(self, info, \*pupil):  for i in pupil:  i.take(info)    class Pupil:  def \_\_init\_\_(self):  self.knowledge = []  def take(self, info):  self.knowledge.append(info)  lesson = Data('class', 'object', 'inheritance', 'polymorphism', 'encapsulation')  marIvanna = Teacher()  vasy = Pupil()  pety = Pupil()  marIvanna.teach(lesson[2], vasy, pety)  marIvanna.teach(lesson[0], pety)  print(vasy.knowledge)  print(pety.knowledge) |

|  |  |  |
| --- | --- | --- |
| **3. Задания** | | |
| 1. | ***Задача:*** | |
|  | Напишите программу по следующему описанию. Есть класс "Воин". От него создаются два экземпляра-юнита. Каждому устанавливается здоровье в 100 очков. В случайном порядке они бьют друг друга. Тот, кто бьет, здоровья не теряет. У того, кого бьют, оно уменьшается на 20 очков от одного удара. После каждого удара надо выводить сообщение, какой юнит атаковал, и сколько у противника осталось здоровья. Как только у кого-то заканчивается ресурс здоровья, программа завершается сообщением о том, кто одержал победу. |
| ***Решение(кодпрограммы):*** | |
|  | import random  class Warrior:  def \_\_init\_\_(self,health):  self.health = health  def hit(attacker,victim):  q = int(input("Enter any button to begin the battle."))  while (warrior1.health > 0) and (warrior2.health > 0):  j = random.randint(1, 2)  if j == 1:  attacker = "Warrior1"  victim = "Warrior2"  warrior2.health -= 20  print(attacker, " has attacked")  print(victim, " now has ", warrior2.health, " HP left\n\n")  else:  attacker = "Warrior2"  victim = "Warrior1"  warrior1.health -= 20  print(attacker, " has attacked")  print(victim, " now has ", warrior1.health, " HP left\n\n")  if (warrior1.health > 0):  print("Warrior1 is today's champion! Warrior2 has turned into a pile of flesh.")  else:  print("Warrior2 is today's champion! Warrior1 has turned into a pile of flesh.")  warrior1 = Warrior(100)  warrior2 = Warrior(100)  hit(warrior1, warrior2) |