Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
«Московский государственный технический университет имени Н.Э. Баумана»  
(МГТУ им. Н.Э. Баумана)  


Факультет \_\_\_\_\_\_\_\_\_ Информатика и системы управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ \_\_

Кафедра \_\_\_\_ Программное обеспечение ЭВМ и информационные технологии\_\_\_\_\_ \_ \_

**Отчет по практике**

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сычев Святослав Антонович\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
(фамилия, имя, отчество)

Группа ИУ7-21 \_\_\_\_\_

Название практики \_\_\_\_ Изучение способов интеграции языков C/C++ и Python\_\_\_ \_\_\_\_\_

Руководитель

Преподаватель Борисов С.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Должность ФИО подпись

Оглавление

[***1.*** ***Введение*** 3](#_Toc455570574)

[***2.*** ***Условная задача*** 3](#_Toc455570575)

[***3.*** ***Способы интеграции*** 4](#_Toc455570576)

[***4.*** ***Решение на Python*** 5](#_Toc455570577)

[***5.*** ***Решение с помощью Ctypes*** 5](#_Toc455570578)

[***6.*** ***Решение с помощью Cython*** 7](#_Toc455570579)

[***7.*** ***Остальные методы интеграции*** 9](#_Toc455570580)

[*7.1.* *SWIG* 9](#_Toc455570581)

[*7.2.* *Boost.Python* 9](#_Toc455570582)

[*7.3.* *Ручное написание интерфейса модуля и пакет distutils* 9](#_Toc455570583)

[***8.*** ***Сравнение скоростей работы методов*** 10](#_Toc455570584)

[***9.*** ***Выводы по различным методам и выбор метода для детального рассмотрения*** 10](#_Toc455570585)

[***10.*** ***Ctypes – способы передачи данных*** 11](#_Toc455570586)

[*10.1.* *Целые и действительные числа* 12](#_Toc455570587)

[*10.2.* *Одномерные массивы* 13](#_Toc455570588)

[*10.3.* *Матрицы* 14](#_Toc455570589)

[*10.4.* *Структуры* 16](#_Toc455570590)

[***11.*** ***Ctypes – считывание из файла в Python и С, сравнение скоростей*** 18](#_Toc455570591)

[*11.1.* *«Чистый» Python 3.4* 19](#_Toc455570592)

[*11.2.* *Использование Ctypes (результат – матрица Ctypes)* 19](#_Toc455570593)

[*11.3.* *Использование Ctypes +numpy (результат – list)* 19](#_Toc455570594)

[*11.4.* *Использование Ctypes +numpy (результат – матрица numpy.array)* 20](#_Toc455570595)

[*11.5.* *Сравнение времен считывания файлов* 20](#_Toc455570596)

[*11.6.* *Вывод* 20](#_Toc455570597)

[***12.*** ***Обобщение полученных результатов*** 21](#_Toc455570598)

[***13.*** ***Литература и полезные ссылки*** 21](#_Toc455570599)

1. ***Введение***

Python – удобная технология для разработки различных типов программных проектов, однако иногда Python в силу своей универсальности, динамической типизации, автоуправления памятью и многих других, без сомнения удобных, особенностей, проигрывает в скорости при использовании «тяжелых» ресурсоемких алгоритмов.   
  
Многие модули и библиотеки Python написаны на C, как следствие этого возникает возможность создавать собственные модули с использованием C/C++ и выносить в них ресурсоемкие части программного продукта.

1. ***Условная задача***

Одно из наиболее эффективных направлений использования интеграции C/C++ и Python – ресурсоемкие задачи. Поэтому в рамках данной работы была разработана условная задача, требующая серьезных затрат ресурсов компьютера.

Работа с большими массивами данных – достаточно ресурсоемкий процесс, особенно, когда приходиться проводить большое число операций.

Теория графов, и задачи поиска оптимальных путей на них зачастую предполагают наличие матрицы, и выполнения большого числа вычислений.

Так как целью данной работы является изучение способов интеграции C/C++ и Python, а не написание оптимального решения достаточно сложной задачи, нам будет достаточно простой, с программной точки зрения, и условной, точки зрения реального применения задачи.

*Задача:*

Некоторая логистическая компания решила оптимизировать процесс выбора пути своих поездов по сети железных дорог, расположенных в достаточно опасной зоне. Часто в этой зоне случаются обвалы, или иные стихийные бедствия, поэтому некоторые пути выходят из строя.

Система датчиков собирает данные об обрывах путей, и отправляет их на главный сервер. Информация об активных путях и времени проезда по ним храниться в виде ориентированного графа, в котором вершины - станции, ребра - Ж/Д пути, вес ребер – время движения по этому пути.

Необходимо написать программу, которая по заданной станции отправления найдет кратчайшее время пути до всех остальных станций.

Граф храниться в файле:  
1 строка – количество вершин  
2 строка – количество ребер   
далее построчно перечислены ребра в виде i j val – ребро из i в j весом val.

Для решения данной задачи будет применен алгоритм Форда-Беллмана, с использованием матрицы смежности графа.

Идея алгоритма:  
1) Рассмотреть каждое ребро i-j и если оно может улучшить время, запомнить это время в массив времен   
2) Рассмотрение ребер повторить N-1 раз, где N – количество вершин.

Алгоритм будет реализован максимально просто, решение будет не оптимальным, однако, его оптимизация усложнит алгоритм и не уменьшит сложность алгоритма в разы, так что, учитывая задачу данной работы, простой реализации будет достаточно.

1. ***Способы интеграции***

Существует несколько альтернативных способов интеграции, часть из них была предусмотрена самими разработчиками Python, часть создали сторонние разработки. Вот основные способы:

* Ручное написание интерфейса модуля
* Пакет distutils
* Пакет Ctypes
* Библиотека Boost.Python
* Проект Cython
* Проект SWIG

Конечным результатом использования каждого из этих способов в большинстве случаев будет библиотека, которую можно будет подключить и использовать в стандартном коде Python.

1. ***Решение на Python***

Первое решение – это решение на «чистом» Python3.4, без использования каких либо дополнительных модулей. Далее приведен листинг алгоритма:



INF – константа условной бесконечности, выше нее время пути быть не может. Она также обозначает отсутствие ребра в матрице смежности, и отсутствие возможности попасть на станцию, если встретиться в соответствующем элементе массива времен.

1. ***Решение с помощью Ctypes***

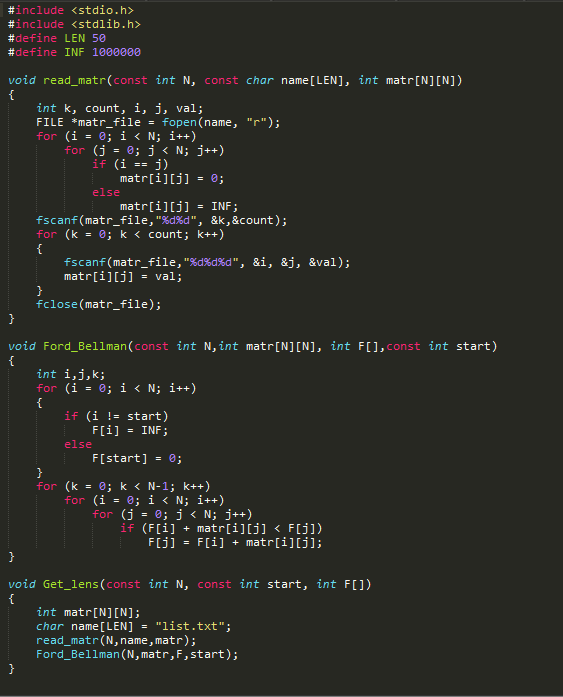
Для работы с этим методом нам потребуется пакет Ctypes из стандартной библиотеки модулей Python (устанавливать пакет не требуется, он поставляется вместе с Python).

Также нам потребуется компилятор C. Далее используется MinGW(gcc), однако можно использовать и другие компиляторы.

Краткое изложение метода:

* Пишем библиотеку на С
* Компилируем ее
* С помощью Ctypes подключаем ее к Python и используем функции из этой библиотеки.

Код библиотеки на C:

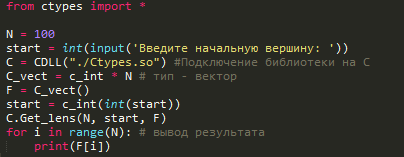


Команда компиляции библиотеки:

gcc -O3 -shared -Wl,-soname,Ctypes.so -o Ctypes.so Ctypes.c

Результатом компиляции является библиотека **Ctypes.so**

Далее полученная библиотека импортируется в программу на Python, в которой нам остается, только задать соответствующий тип массива, используя Ctypes, и вызвать функцию Get\_lens, передав ей этот массив.  
  
Листинг программы на Python:



Стоит отметить, что в данной работе рассмотрено только частное применение Ctypes, его можно использовать даже без создания библиотеки на С, а также он позволяет создавать гораздо более серьезные библиотеки, со «сшитыми» структурами, передающимися между кодом Python и библиотекой.

1. ***Решение с помощью Cython***

Cython – отдельный программный продукт, позволяющий компилировать код Python и позволяющий интегрировать в этот код функции из библиотек С.

Cython можно установить при помощи pip - системы управления пакетами, Python.

pip поставляется вместе с Python начиная с версии 3.4, если ваша версия python меньше, то его необходимо установить отдельно (способы установки pip описаны на официальном сайте [9])

Для установки Cython необходимо в командной строке ввести команду:

pip install cython

Если система не знает команды pip, необходимо прописать в переменных средах PATH путь к директории …Python…\Tools\Scripts\.

В данной работе будет рассмотрен пример компиляции «чистого» кода Python, без добавления функций на С, так как в Cython без проблем импортируются функции из базовых библиотек С, однако в ходе выполнения работы при попытке создания сторонней библиотеки, и импортирования функций из нее возникли проблемы, было принято решение рассмотреть на сколько компиляция «чистого» кода ускорит Python.

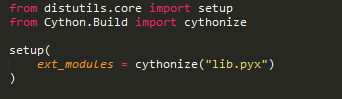
Общий принцип работы Cython:

* Создается файл-pyx на языке Cython
* Создается конфигурационный файл – setup.py
* Далее выполняется компиляция:  
  Сначала Cython компилирует .pyx-файл в .с-файл, после чего компилятор C компилирует .с-файл в .so-файл, который можно импортировать, как обычный модуль Python

Листинг .pyx-файла:



Листинг конфигурационного файла:



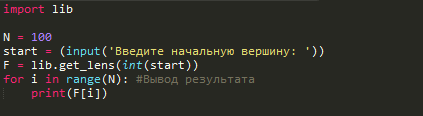
Далее проводим компиляцию:

python setup.py build\_ext --inplace --compiler=mingw32

Примечание – использован компилятор mingw32 (свободно распространяемый)

Результатом компиляции будет библиотека **lib.so**

Далее подключаем эту библиотеку к программе на Python, и вызываем функцию get\_lens



1. ***Остальные методы интеграции***

В данной работе остальные методы будут представлены обзорно, так как они оказались довольно сложны в установке и настройке (3 дня рабочего времени было потрачено на установку и настройку SWIG и Boost.Python, однако ни один из проектов так и не дал положительных результатов).

* 1. *SWIG*

SWIG - [свободный](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) инструмент для [связывания](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B2%D1%8F%D0%B7%D1%8B%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D1%8F%D0%B7%D1%8B%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)&action=edit&redlink=1) [программ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) и [библиотек](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), написанных на [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))/[C++](https://ru.wikipedia.org/wiki/C%2B%2B), с [интерпретируемыми](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80), такими как Python, Perl, Ruby, или [компилируемыми](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) – Java, C#, языками.

Связывание происходит посредством создания специального .i-файла на специальном языке SWIG, после чего SWIG автоматически генерирует исходный код на C, после компиляции которого, можно получить библиотеку.

* 1. *Boost.Python*

Boost представляет собой **набор** из нескольких десятков разнообразных инструментов от **различных** разработчиков, иногда очень слабо связанных между собой.

Инструмент для Python - Boost Python Library, или сокращенно Boost.Python

Принцип работы Boost.Python заключается в следующем:

* Пишется исходный код библиотеки на C (на С++)
* Создается общий заголовочный файл
* Создается Файл интерфейса Boost (на С++)
* После чего модуль собирается при помощи специальных средств Boost, или вручную.

В итоге получается библиотека для Python.

К сожалению, Boost – устаревший продукт, так как его разработка завершена к 2003 году, корректность его работы с Python 3 под большим вопросом.

* 1. *Ручное написание интерфейса модуля и пакет distutils*

Этот метод подразумевает прописывание всех интерфейсов вручную, что требует глубокого изучения Python\C API, для получения достойного результата.

Пакет distutils немного упрощает этот процесс, позволяя писать файл интерфейса не на C, а на Python, однако в этом случае также требуется знание Python\C API, для написания корректно работающей библиотеки.

1. ***Сравнение скоростей работы методов***

Далее приведены результаты замеров среднего времени реализаций на «чистом» Python, Python + Ctypes и Python + Cython. Для более точного замера времени производиться многократный запуск, после чего берется среднее время работы.

Графы в каждом замере для всех реализаций одинаковы

Время приведено в миллисекундах.

|  |  |  |  |
| --- | --- | --- | --- |
| N – вершин | Python3.4 | Python3.4 + Ctypes | Python3.4 + Cython |
| 100 | 324.13 | 5.45 | 191.43 |
| 200 | 2535.14 | 23.47 | 1346.36 |
| 300 | 8721.04 | 55.04 | 4666.32 |
| 400 | 20267.42 | 115.08 | 11135.89 |
| 500 | 39041.78 | 198.86 | 21955.66 |

Примечание:

Для N = 400 и 500 реализация на Python3.4 была запущена по 1 разу, для экономии времени в виду явной длительности выполнения (погрешность будет мала)

Для создания графов была написана специальная программа на языке Python.

1. ***Выводы по различным методам и выбор метода для детального рассмотрения***

В ходе выполнения работы мною были рассмотрены все вышеперечисленные методы.

К сожалению SWIG и Boost.Python оказались очень сложны в установке и настройке (до конца их настроить так и не удалось).

Ручное написание интерфейсов и пакет distutils очень объемные методы, сложные в использовании из-за необходимости дополнительного изучения Python\C API.

Cython позволяет ускорять код написанный на «чистом» Python или использовать функции из стандартных библиотек C.

Наиболее простой и эффективной в рамках данной работы оказалась реализация на Сtypes. Ускорение работы при выносе всех затратных операций в библиотеку на C оказалось очень значительным (примерно в 200 раз).

Далее будет более подробно рассмотрен модуль Ctypes, как наиболее эффективный простой в использовании.

1. ***Ctypes – способы передачи данных***

Рассмотрим более подробно способы передачи различных типов данных из программы на Python в функции библиотеки на C и обратно.

В Ctypes есть множество типов даных, для интеграции с библиотеками и функциями на C.

Целочисленные типы:

|  |  |  |
| --- | --- | --- |
| Тип Ctypes | Соответствующий тип C | Соответствующий тип Python |
| [c\_byte](https://docs.python.org/2/library/ctypes.html#ctypes.c_byte) | char | int/long |
| [c\_ubyte](https://docs.python.org/2/library/ctypes.html#ctypes.c_ubyte) | unsigned char | int/long |
| [c\_short](https://docs.python.org/2/library/ctypes.html#ctypes.c_short) | short | int/long |
| [c\_ushort](https://docs.python.org/2/library/ctypes.html#ctypes.c_ushort) | unsigned short | int/long |
| [c\_int](https://docs.python.org/2/library/ctypes.html#ctypes.c_int) | int | int/long |
| [c\_uint](https://docs.python.org/2/library/ctypes.html#ctypes.c_uint) | unsigned int | int/long |
| [c\_long](https://docs.python.org/2/library/ctypes.html#ctypes.c_long) | long | int/long |
| [c\_ulong](https://docs.python.org/2/library/ctypes.html#ctypes.c_ulong) | unsigned long | int/long |
| [c\_longlong](https://docs.python.org/2/library/ctypes.html#ctypes.c_longlong) | \_\_int64 or long long | int/long |
| [c\_ulonglong](https://docs.python.org/2/library/ctypes.html#ctypes.c_ulonglong) | unsigned \_\_int64 or unsigned long long | int/long |

Действительные типы:

|  |  |  |
| --- | --- | --- |
| Тип Ctypes | Соответствующий тип C | Соответствующий тип Python |
| [c\_float](https://docs.python.org/2/library/ctypes.html#ctypes.c_float) | float | float |
| [c\_double](https://docs.python.org/2/library/ctypes.html#ctypes.c_double) | double | float |
| [c\_longdouble](https://docs.python.org/2/library/ctypes.html#ctypes.c_longdouble) | long double | float |

Далее будет использоваться тип c\_int и c\_float, как наиболее часто используемые.

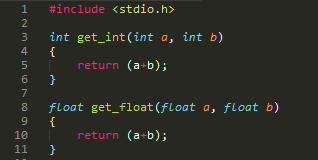
Больше информации о типах Ctypes можно найти в официальной документации [1]

Стоит отметить, что при использовании Ctypes зачастую может происходить приведение типов (что достаточно удобно).

* 1. *Целые и действительные числа*

Работа с целочисленными и действительными типами данных наиболее проста.

Код библиотеки (lib.c)



В этом примере библиотека содержит две идентичные функции для работы с целыми и действительными числами.

Код Python (example1.py)



В 3 строке подключается библиотека (lib.so), расположенная в той же директории, что и файл example.py.

В случае целых чисел, зачастую можно не приводить переменные к соответствующему Ctypes типу, это произойдет автоматически, если же ошибки все же возникнут, необходимо поступить так же, как и в случае с действительными – преобразовать типы «вручную».

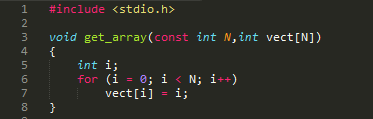
Также при работе с действительными числами необходимо указать возвращаемый тип функции (31 строка). В нашем случае это c\_float.

* 1. *Одномерные массивы*

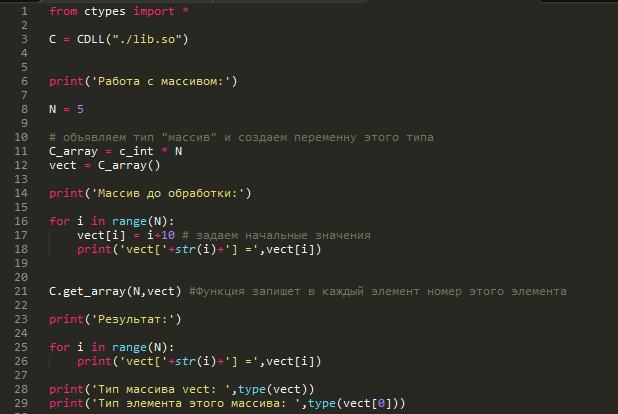
При работе с массивами нет возможности передать в С стандартный список Python, так как это динамический массив, с множеством дополнительных функций.

Для передачи или приема массива из функции на С в Ctypes можно создать свой тип данных (массив необходимого размера).

Код библиотеки (lib.c)



Код Python (example2.py)



В библиотеке реализована функция, принимающая на вход размер массива и сам массив, и изменяющая этот массив. Такой способ «возвращения» массива наиболее удобен при работе с Сtypes, хотя, возможны и другие реализации.

В коде Python также необходимо подключить библиотеку (3 строка), в последующих примерах эта строка комментироваться не будет.

Далее мы определяем свой собственный тип C\_array (11 строка), состоящий из N элементов типа c\_int.

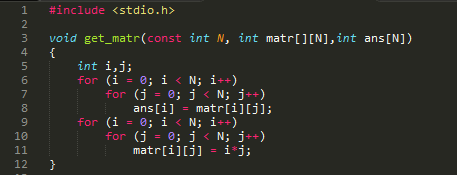
После этого определяется тип переменной vect (12 строка)

После этого достаточно передать этот массив в функцию, и после ее вызова, значения элементов массива изменяться. Таким образом, можно передавать и получать массивы из функций на C.

* 1. *Матрицы*

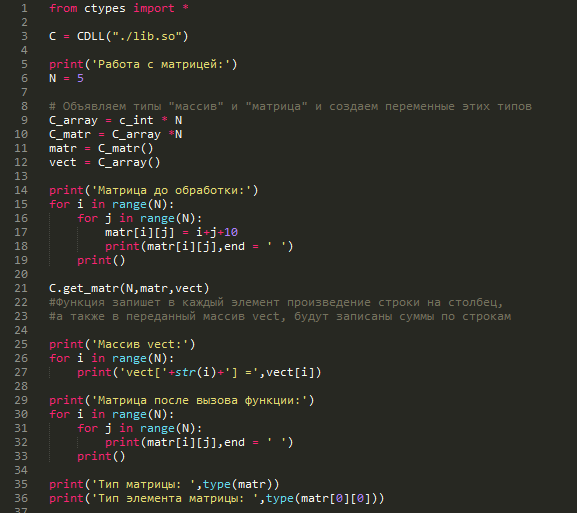
Работа с матрицами похожа на работу с массивами, только здесь необходимо определить уже два типа – массив и матрицу (составленную из массивов).

Код библиотеки (lib.c)



В библиотеке реализована функция, получающая на вход размерность, матрицу и массив, и меняющая их значения.

Код Python (example3.py)



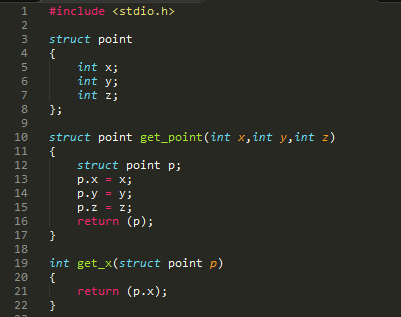
Нового, по сравнению с примером 2 здесь не много, только объявление второго типа (10 строка).

К элементам матрицы можно обращаться как к обычным элементам многомерного массива.

* 1. *Структуры*

В Ctypes существует возможность создать единую c библиотечной структуру.

Код библиотеки (lib.c)



В библиотеке реализована структура point, задающая точку в Евклидовом пространстве, а также две функции:

Первая получает на вход три координаты и возвращает структуру.

Вторая – получает на вход структуру и возвращает x координату точки в пространстве.

Для того, чтобы структуры можно было беспрепятственно передавать между библиотекой и кодом Python, необходимо в коде Python описать структуру Ctypes, в точности соответствующую структуре из библиотеки (совпадение названия, полей и их типов).

Описание такой структуры содержиться в строках 9-10.

Поля структуры описываются как массив кортежей из названия и типа поля.

Код Python (example4.py)



1. ***Ctypes – считывание из файла в Python и С, сравнение скоростей***

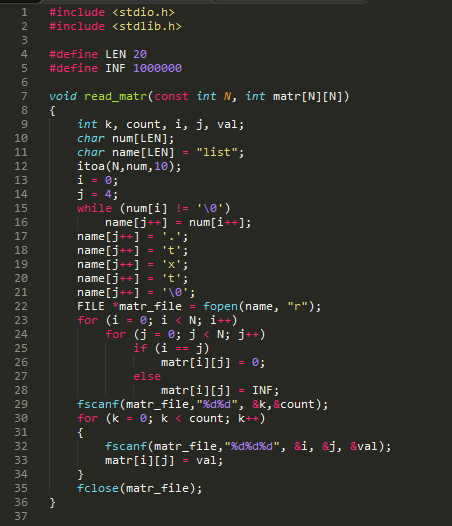
В этом разделе приведены различные способы считывания информации из файла.

Для этого исследования были использованы файлы и часть библиотеки из примера реализации условной задачи с помощью Ctypes.

Реализованы были следующие методы:

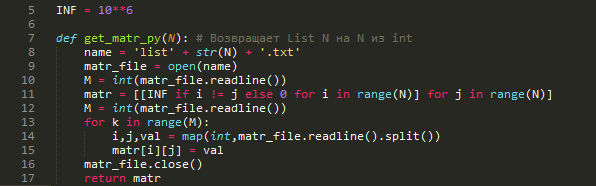
* + «Чистый» Python 3.4 (результат – list)
  + Использование Ctypes (результат – матрица Ctypes)
  + Использование Ctypes +numpy (результат – list)
  + Использование Ctypes +numpy (результат – матрица numpy.array)

Код библиотеки:



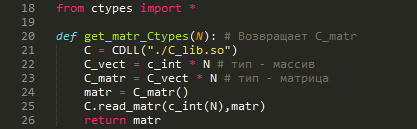
* 1. *«Чистый» Python 3.4*

Этот метод будет понятен любому, кто умеет программировать на Python, поэтому комментарии тут излишни:



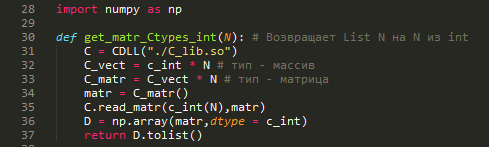
* 1. *Использование Ctypes (результат – матрица Ctypes)*

Этот метод сводиться к объявлениям функций Ctypes и вызову функции из библиотеки:



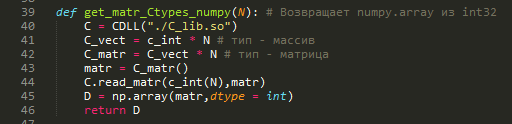
* 1. *Использование Ctypes +numpy (результат – list)*

В этом методе были использованы возможности пакета numpy, для быстрого преобразования C\_matr в list.



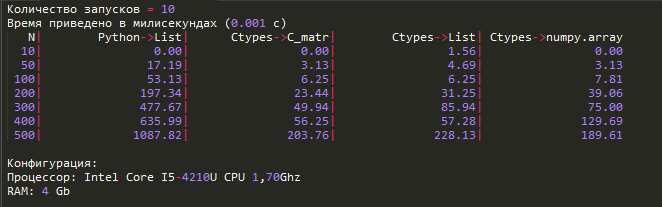
* 1. *Использование Ctypes +numpy (результат – матрица numpy.array)*

В четвертый метод, полностью копирует третий, но в нем возвращается просто матрица numpy.array.



* 1. *Сравнение времен считывания файлов*

Полный код программы включает в себя процедуры замера времени работы, и форматный вывод результатов:



Каждый метод для повышения точности на каждом массиве был запущен по 10 раз, после чего взято среднее время.

* 1. *Вывод*

Время считывания данных из файла при помощи Ctypes и библиотеки на С\С++ в разы превышает время считывания «чистого» Python 3.4, при этом время выполнения трех вариантов с использованием Ctypes сравнимо, на основе этого можно делать вывод:

Если файл достаточно большой, и информацию из него удобно считывать при помощи функции на С\С++, **то лучше это делать** (применяя Ctypes).

1. ***Обобщение полученных результатов***

Итак, в ходе данной работы были рассмотрены различные методы интеграции Python и С\С++. Было выявлено, что такая интеграция возможна, и при решении «тяжелых» по памяти и времени задач дает ощутимый прирост скорости.

Наиболее удобным методом интеграции (по совокупности простоты и удобства установки и применения) был выбран пакет Ctypes.

Для этого пакета приведены основные способы передачи данных, а также проведено исследование времени работы с файлами.

В ходе выполнения работы было реализовано несколько библиотек (shared library), написана программа для автоматической генерации файлов к условной задаче, а также три реализации решения условной задачи с применением различных методов интеграции.

Для хранения файлов работы был использован Git – репозиторий [8] Все исходные коды, библиотеки и файлы, использованные и описанные в этой работе можно найти там.

1. ***Литература и полезные ссылки***
2. Документация к Cytpes: [https://docs.python.org/2/library/ctypes.html#](https://docs.python.org/2/library/ctypes.html)
3. Документация к Cython: <http://docs.cython.org>
4. Документация к SWIG: <http://www.swig.org/doc.html>
5. Документация к Boost.Python: <http://www.boost.org/doc/libs/1_58_0/libs/python/doc/>
6. Документация к distutils: <https://wiki.python.org/moin/Distutils>
7. Статья по интеграции Python и С\C++: <http://www.ibm.com/developerworks/ru/library/l-python_details_06/index.html>
8. «Язык программирования С» [Керниган Б.](http://mexalib.com/author/%D0%9A%D0%B5%D1%80%D0%BD%D0%B8%D0%B3%D0%B0%D0%BD%20%D0%91.) [Ритчи Д.](http://mexalib.com/author/%D0%A0%D0%B8%D1%82%D1%87%D0%B8%20%D0%94.) издательство Вильямс 2-e издание
9. Ссылка на репозиторий с файлами работы: <https://github.com/Sych474/Practic_in_BMSTU/>
10. Установка pip <https://pip.pypa.io/en/stable/installing/>