

Московский государственный университет имени М.В.Ломоносова

Университет МГУ-ППИ в Шэньчжэне

Факультет вычислительной математики и кибернетики

Разработка компонентов электронного задачника, связанных с обработкой структур данных

Цюй Жуйчэнь

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:

к.ф-м.н., доцент

М. Э. Абрамян

Шэньчжэнь, 2024

Аннотация

Разработка компонентов электронного задачника, связанных с обработкой структур данных

Цюй Жуйчэнь

Работа посвящена реализации новой кроссплатформенной версии электронного задачника Programming Taskbook. Рассмотрена архитектура новой версии и ее отличия от предыдущих версий. Описан вариант конструктора учебных заданий для языка C++. Обсуждаются особенности адаптации компонентов задачника, реализованных для ОС Windows, к ОС Linux. Приведены примеры использования задачника в ОС Linux, в частности, даны образцы решений задач на обработку структур данных из стандартной библиотеки шаблонов языка C++ и задач на реализацию параллельных матричных алгоритмов на основе технологии MPI.

Содержание

Введение	3
Постановка задачи	5
1 Общее описание задачника РТ5	6
1.1 Отличия от РТ4 и UnixTaskbook	6
1.2 Новые возможности	7
2 Конструктор для языка C++	9
2.1 Описание структуры конструктора для C++	9
2.2 Описание возможностей конструктора для C++	12
3 Перенос компонентов РТ5 и конструктора в Linux	17
3.1 Особенности переноса в Linux	17
3.2 Проблемы, возникшие при переносе	21
4 Примеры работы задачника и конструктора для Linux	23
4.1 Примеры работы задачника для Linux	23
4.2 Демонстрационная группа для конструктора C++	26
4.3 Реализация для задачника РТ5 групп заданий по изучению библиотеки STL	31
4.4 Реализация для задачника РТ5 группы заданий по изучению параллельных матричных алгоритмов	34
Заключение	41
Список литературы	42

Введение

В работе описываются компоненты новой версии электронного задачника по программированию Programming Taskbook. Существующая версия 4 была реализована для ОС Windows [1]. Новая версия 5 поддерживает как ОС Windows, так и ОС семейства Unix, и включает тот же базовый набор учебных заданий. Особенностью новой версии является более простой механизм взаимодействия учебных программ и ядра задачника, а также пользовательский интерфейс, основанный на выводе в консоль, что позволяет применять задачник в операционных системах без графического интерфейса.

Первые результаты, связанные с разработкой кроссплатформенной версии электронного задачника, были получены в 2020—2021 гг., когда был осуществлен перенос ядра задачника с платформы Delphi на платформу Lazarus (www.lazarus-ide.org) [2, 3], что позволило реализовать его новый 64-разрядный вариант и тем самым обеспечить интеграцию с 64-разрядными системами программирования (в частности, для языков Python, Java, C). Ко времени выпуска версии 2.22 (2022 г.) была достигнута полная унификация графического интерфейса для 32- и 64-разрядного варианта, а также выполнена адаптация всех специализированных расширений задачника к 64-разрядному варианту. В этой версии была реализована расширенная поддержка Юникода, что обеспечило успешное применение задачника в учебных курсах, проводимых в Университете МГУ-ППИ в Шэньчжэне (КНР).

Компоненты для поддержки заданий из задачника РТ4, реализованные в задачнике Unix Taskbook, включали базовые динамические библиотеки, созданные в системе Lazarus на основе исходных модулей с заданиями для задачника РТ4, а также библиотеки-адаптеры, предназначенные для адаптации исходного интерфейса библиотек РТ4 к интерфейсу задачника Unix Taskbook. Эти библиотеки-адаптеры, как и ядро задачника Unix Taskbook, были реализованы на языке C++. Однако особенности интерфейса задачника Unix Taskbook не позволили в полной мере реализовать для него все возможности, имеющиеся в задачнике РТ4. Кроме того, использование в ядре задачника Unix Taskbook ряда библиотек языка C++ для ОС Unix препятствовало его переносу в систему Windows.

Поэтому возникла идея реализовать новую кроссплатформенную версию задачника Programming Taskbook, сходную по возможностям с задачиком Unix Taskbook, на платформе Lazarus. Выбор среды Lazarus был обусловлен тем, что для этой среды уже были реализованы все модули с заданиями, конструктор заданий, а также

дополнительные модули, входящие в РТ4, которые можно было бы с минимальными модификациями включить в новую версию. Благодаря такому подходу упростилась и архитектура, так как отпала необходимость в специальных библиотеках-адаптерах. В то же время была сохранена важная возможность всех прежних версий задачника Programming Taskbook, связанная с использованием специальных средств ввода-вывода и отладки, которые существенно упрощают процесс решения задач.

После разработки прототипа задачника Programming Taskbook 5 для ОС Windows, в который были перенесены все основные элементы архитектуры, ранее реализованные в ядре задачника Unix Taskbook, возникла необходимость в дальнейшем расширении возможностей новой версии. На первом этапе было расширено семейство конструкторов учебных заданий для версии 5: имеющийся конструктор для языка Free Pascal был дополнен конструктором для языка C++. Это позволило перенести в новую версию задачника группы учебных заданий, реализованные на языке C++ в предыдущей версии, а именно группы, связанные с изучением библиотеки STL языка C++ и технологии распределенного программирования MPI для языков C/C++. На следующем этапе версия 5 была адаптирована для платформы Linux. Ядро новой версии, как и предыдущей, разработано на языке Free Pascal в мультиплатформенной среде Lazarus. Была также выполнена адаптация для платформы Linux конструктора учебных задания для языка C++ и всех групп заданий, разработанных с применением этого конструктора.

Результатом работы является полнофункциональная мультиплатформенная версия задачника Programming Taskbook, которая включает более 2000 учебных заданий и позволяет их выполнять на языках C и C++.

Постановка задачи

Реализовать расширенный кроссплатформенный вариант электронного задачника Programming Taskbook версии 5. Для этого:

- В дополнение к существующему конструктору учебных заданий для языка Free Pascal разработать вариант конструктора для языка C++;
- С помощью разработанного конструктора перенести в новую версию задачника существующие группы заданий, связанные с изучением структур данных, входящих в стандартную библиотеку шаблонов языка C++, а также с изучением технологии параллельного программирования MPI;
- Адаптировать к ОС Linux все компоненты задачника Programming Taskbook версии 5, реализованные для ОС Windows.

1 Общее описание задачника PT5

Прототип новой версии электронного задачника был разработан в рамках выполнения выпускных работ в 2022-23 учебном году. Особенностью этой версии является ориентация на консольный вывод, что не требует применения графической оболочки и упрощает адаптацию этой версии к другим операционным системам. В имеющемся прототипе была реализована новая архитектура, обеспечивающая более простое взаимодействие учебных программ и ядра задачника. При этом было сохранено разделение задачника на ядро и комплекс подключаемых динамических библиотек, содержащих реализации групп учебных заданий. Был разработан новый вариант конструктора учебных заданий, с помощью которого удалось легко адаптировать ранее разработанные группы заданий к новой архитектуре. Созданный прототип позволял выполнять задания базового уровня, а также начальные группы заданий из задачника по параллельному программированию для языков C и C++.

1.1 Отличия от PT4 и UnixTaskbook

Задачник Programming Taskbook 5 (PT5) обладает рядом важных отличий как от своей предыдущей версии PT4, так и от Unix Taskbook. Одно из ключевых отличий от PT4 заключается в использовании консольного интерфейса для отображения результатов выполнения заданий, тогда как в PT4 для этих целей использовалось специальное окно с графическим интерфейсом. Переход к консольному интерфейсу позволил использовать задачник в операционных системах Linux, не имеющих графического интерфейса, что в свою очередь существенно упрощает перенос задачника на другие платформы. Современные возможности консолей Windows и Linux обеспечивают высокий уровень наглядности, аналогичный версии PT4, включая использование псевдографики и цветовых схем. Кроме того, PT5 поддерживает вывод отладочной информации и дополнительной информации о задании с применением кодировки UTF-8, что позволяет включать в текст любые символы Юникода.

Еще одним важным отличием от PT4 является использование единой управляющей программы PT5Run, которая выполняет все функции, связанные с созданием заготовки учебной программы, ее компиляцией и запуском, а также отображением результатов. В отличие от PT4, где средства ввода-вывода импортировались учебной программой непосредственно из ядра задачника, в PT5 ядро не взаимодействует напрямую с учебной программой при генерации данных задания, а сохраняет инфор-

мацию на диск в специальном формате. Учебная программа считывает эти данные и сохраняет результаты самостоятельно, что упрощает реализацию ядра задачника и позволяет легче адаптировать задачник к новым языкам программирования.

В сравнении с Unix Taskbook, PT5 предлагает кроссплатформенную поддержку, реализованную на платформе Lazarus, что значительно облегчает его переносимость на различные операционные системы, в отличие от Unix Taskbook, который сильно зависел от библиотек C++ для Unix и был сложно переносим в Windows. Кроме того, в PT5 устранена необходимость в специальных библиотеках-адаптерах, что упрощает архитектуру задачника. Как и в Unix Taskbook, в PT5 используется единая управляющая программа PT5Run, но в PT5 она обеспечивает более простой и удобный интерфейс для взаимодействия с учебными программами.

1.2 Новые возможности

Задачник Programming Taskbook 5 реализован как для ОС Windows, так и для ОС Linux, что обеспечивает его использование в различных операционных системах. Одной из ключевых новых возможностей является использование современного консольного интерфейса, который позволяет отображать результаты выполнения заданий с помощью псевдографики и различных цветовых схем. Поддержка UTF-8 позволяет включать любые символы Юникода, что значительно расширяет возможности по отображению информации.

Простое взаимодействие учебных программ с ядром задачника достигнуто за счет того, что учебная программа и управляющая программа PT5Run являются независимыми приложениями, которые просто запускаются одно из другого. Это упрощает взаимодействие и облегчает адаптацию задачника к новым языкам программирования. PT5 также поддерживает выполнение заданий на языках C и C++ с использованием MPI, что позволяет запускать учебные программы через управляющую программу mpiexec для параллельных вычислений.

Значительно расширены возможности настройки задачника: учащийся может изменить вид отображения результатов, цветовую схему и другие параметры непосредственно в консольном окне с помощью простых управляющих команд. Кроме того, PT5 позволяет представлять информацию, связанную с заданием, в виде HTML-документов.

Разработан конструктор PT5Make для языка C++, что обеспечило перенос в задачник PT5 оставшихся групп заданий из PT4, включая завершающую группу

MPI9Matr из расширения PT4 for MPI-2 и все группы из расширения задачника PT4 по изучению библиотеки STL.

Таким образом, в настоящее время задачник PT5 включает практически все группы заданий из базового набора PT4 (за исключением задач, связанных с обработкой динамических структур данных) и все группы заданий из расширений задачника, посвященных стандартной библиотеке шаблонов C++, параллельному MPI-программированию и паттернам проектирования.

2 Конструктор для языка C++

2.1 Описание структуры конструктора для C++

Во-первых, в разделе «Определения типов функций динамической библиотеки» мы определяем ряд типов указателей функций, которые используются для ссылок на функции в динамической библиотеке. Эти определения включают в себя типы функций без параметров и с параметрами, такие как TNFunc, TNFuncN4, TSFunc, TSFuncN, TProc и другие. Благодаря этим определениям мы можем гибко вызывать функции из динамической библиотеки, не заботясь о деталях их реализации.

```
typedef int      (_stdcall *TNFunc)(void);
typedef int      (_stdcall *TNFuncN4)(int, int, int, int);
typedef char*    (_stdcall *TSFunc)(void);
typedef char*    (_stdcall *TSFuncN)(int);
typedef void     (_stdcall *TProc)(void);
typedef void     (_stdcall *TProcS)(const char *);
typedef void     (_stdcall *TProcSCN2)(const char *, char, int, int);
typedef void     (_stdcall *TProcSN)(const char *, int);
typedef void     (_stdcall *TProcSN2)(const char *, int, int);
typedef void     (_stdcall *TProcSN3)(const char *, int, int, int);
typedef void     (_stdcall *TProcSN4)(const char *, int, int, int, int);
typedef void     (_stdcall *TProcSN5)(const char *, int, int, int, int, int);
typedef void     (_stdcall *TProcSN6)(const char *, int, int, int, int, int, int);
typedef void     (_stdcall *TProcSRN3)(const char *, double, int, int, int);
typedef void     (_stdcall *TProcSR2N3)(const char *, double, double, int, int, int);
typedef void     (_stdcall *TProcSR3N3)(const char *, double, double, double, int,
                                         int, int);

typedef void     (_stdcall *TProcS2)(const char *, const char *);
typedef void     (_stdcall *TProcS2N2)(const char *, const char *, int, int);
typedef void     (_stdcall *TProcS4NP)(const char *, const char *, const char *,
                                         const char *, int, void *);

typedef void     (_stdcall *TProcN)(int);
typedef void     (_stdcall *TProcNP)(int, void *);
typedef void     (_stdcall *TProcN2)(int, int);
typedef void     (_stdcall *TProcN3)(int, int, int);
```

```

typedef void    (_stdcall *TProcN4)(int, int, int, int);
typedef void    (_stdcall *TProcSvN)(const char *, int *);
typedef void    (_stdcall *TProcSNS)(const char *, int, const char *);
...

```

Во-вторых, в разделе реализации класса мы определяем класс PT5TaskMakerLink и объявляем в нем несколько переменных-членов указателей функций, таких как creategroup, usetask, createtask и другие. Эти переменные-члены загружаются функцией GetProcAddress для получения адреса конкретной функции в динамической библиотеке. Для обеспечения корректной загрузки, если адрес функции не найден, выдается сообщение об ошибке. Эта часть реализации гарантирует, что функции динамической библиотеки могут быть вызваны из класса для реализации определенной функциональности.

```

PT5TaskMakerLink PTLINK;

void * GetProcAddress(HMODULE handle, const char *s)
{
    void *p = GetProcAddress(handle, s);
    if (p == 0)
        std::cout << "Not found: " << s << '\n';
    return p;
}

void PT5TaskMakerLink::activate(const char* DllName) {
    FHandle = LoadLibraryA(DllName);
    creategroup_ = (TProcS4NP) GetProcAddress(FHandle, "creategroup");
    usetask_ = (TProcSN) GetProcAddress(FHandle, "usetask");
    createtask_ = (TProcS) GetProcAddress(FHandle, "createtask");
    currentlanguage_ = (TNFunc) GetProcAddress(FHandle, "currentlanguage");
    tasktext_ = (TProcSN2) GetProcAddress(FHandle, "tasktext");
    datab_ = (TProcSN3) GetProcAddress(FHandle, "datab");
    datan_ = (TProcSN4) GetProcAddress(FHandle, "datan");
    datar_ = (TProcSRN3) GetProcAddress(FHandle, "datar");
    datacomment_ = (TProcSN2) GetProcAddress(FHandle, "datacomment");
    resultb_ = (TProcSN3) GetProcAddress(FHandle, "resultb");
    resultn_ = (TProcSN4) GetProcAddress(FHandle, "resultn");
    resultr_ = (TProcSRN3) GetProcAddress(FHandle, "resultr");
}

```

```

resultcomment_ = (TProcSN2) GetProcAddress(FHandle, "resultcomment");
setprecision_ = (TProcN) GetProcAddress(FHandle, "setprecision");
settestcount_ = (TProcN) GetProcAddress(FHandle, "settestcount");
setrequireddatacount_ = (TProcN) GetProcAddress(FHandle,
"setrequireddatacount");
center_ = (TNFuncN4) GetProcAddress(FHandle, "center");
datafilen_ = (TProcSN2) GetProcAddress(FHandle, "datafilen");
datafiler_ = (TProcSN2) GetProcAddress(FHandle, "datafiler");
datafilec_ = (TProcSN2) GetProcAddress(FHandle, "datafilec");
resultfilen_ = (TProcSN2) GetProcAddress(FHandle, "resultfilen");
resultfiler_ = (TProcSN2) GetProcAddress(FHandle, "resultfiler");
resultfilec_ = (TProcSN2) GetProcAddress(FHandle, "resultfilec");
cleargroupdata_ = (TProc) GetProcAddress(FHandle, "cleargroupdata");
gettopic_ = (TProcvS) GetProcAddress(FHandle, "gettopic");
gettaskcount_ = (TNFunc) GetProcAddress(FHandle, "gettaskcount");
settaskdata_ = (TProcN2vS) GetProcAddress(FHandle, "settaskdata");
gettestcount_ = (TNFunc) GetProcAddress(FHandle, "gettestcount");
getcode_ = (TProcvS) GetProcAddress(FHandle, "getcode");
setlanguage_ = (TProcN) GetProcAddress(FHandle, "setlanguage");
pause_ = (TProc) GetProcAddress(FHandle, "pause");
randomn_ = (TNFuncN2) GetProcAddress(FHandle, "randomn");
tasktext0_ = (TProcS) GetProcAddress(FHandle, "tasktext0");
setgroupdata_ = (TProcS4N) GetProcAddress(FHandle, "setgroupdata");
...
}

```

В среде Windows метод activate класса PT5TaskMakerLink реализует функцию динамической загрузки DLL-файла и получения адресов экспортируемых функций в нем. В частности, метод сначала использует функцию LoadLibraryA для загрузки указанного DLL-файла, передает имя DLL-файла для получения хэндла модуля этого файла и сохраняет его в переменной-члене FHandle класса. Если загрузка не удалась, FHandle будет равна nullptr.

Далее метод использует функцию-обертку GetProcAddress для получения адресов ряда функций из загруженного модуля DLL. Функция GetProcAddress вызывает функцию Windows API GetProcAddress, которая возвращает имя указанной

функции, если она существует в DLL, в противном случае возвращается nullptr. GetProcAddress возвращает адрес функции, если указанное имя функции существует в DLL, в противном случае возвращает nullptr. Кроме того, GetProcAddress1 выводит сообщение об ошибке, если адрес функции не найден, для целей отладки и устранения неполадок.

Полученный адрес функции преобразуется в предопределенный тип указателя функции и сохраняется в переменной-члене класса. Например, creatategroup_ содержит адрес функции creategroup, usetask_ содержит адрес функции usetask, а createtask_ содержит адрес функции createtask. Аналогично, остальные переменные-члены указателей функций назначаются на соответствующие адреса функций. Таким образом, экземпляры класса PT5TaskMakerLink могут вызывать функции в DLL через эти указатели функций во время выполнения программы.

2.2 Описание возможностей конструктора для C++

CreateGroup принимает имя подгруппы в качестве аргумента при создании задачи, которое указывается первым при определении новой задачи. Основная функция заключается в создании группы задач и инициализации генератора случайных чисел. Создание группы задач основывается на передаче различной информации о параметрах и функции обратного вызова для инициализации задачи. Затем инициализируется генератор случайных чисел, используя текущее время в качестве затравки, что обеспечивает изменение случайного поведения программы с течением времени, добавляя динамичности и непредсказуемости программе.

```
inline void CreateGroup(const char* GroupName, const char*
GroupDescription, const char* GroupAuthor, const char* GroupKey,
int TaskCount, TInitTaskProc InitTaskProc);
```

UseTask позволяет импортировать существующую группу задач в новую группу задач. После импорта изменится только название задачи и описание группы, остальное останется прежним.

```
inline void UseTask(const char* GroupName, int TaskNumber);
```

Эти три перегруженные функции CreateTask позволяют гибко передавать параметры при создании задач и групп подзадач, либо предоставляя имя группы под-

задач независимо, либо предоставляя как имя группы подзадач, так и количество обрабатываемых задач, либо предоставляя только количество обрабатываемых задач. Такой подход повышает модульность и расширяемость системы и может быть адаптирован к потребностям различных сценариев применения.

```
inline void CreateTask(const char* SubgroupName, int*);  
inline void CreateTask(const char* SubgroupName);  
inline void CreateTask(int *);
```

Основная задача функции TaskText - задать текстовое содержимое задания, при этом вы можете указать координаты (X, Y) расположения текста или нет.

```
inline void TaskText(const char* S, int X , int Y);  
inline void TaskText(const char* S);
```

Эти методы предназначены для добавления к исходным данным различных типов данных (булевых, целочисленных, вещественных, символьных и строковых) с комментариями и координатами. Они добавляют переданные параметры к исходным данным.

```
inline void DataB(const char* Cmt, bool B, int X, int Y);  
inline void DataN(const char* Cmt, int N, int X, int Y, int W);  
inline void DataN(const char* Cmt, int N1, int N2, int X, int Y, int W);  
inline void DataN(const char* Cmt, int N1, int N2, int N3, int X,int Y,int W);  
inline void DataR(const char* Cmt, double R, int X, int Y, int W);  
inline void DataR(const char* Cmt, double R1, double R2, int X,int Y,int W);  
inline void DataR(const char* Cmt, double R1, double R2,double R3,int X,int Y,  
int W);  
inline void DataC(const char* Cmt, char C, int X, int Y);  
inline void DataS(const char* Cmt, const char* S, int X, int Y);  
inline void DataComment(const char* Cmt, int X, int Y);
```

Эти методы предназначены для добавления к исходным данным различных типов результатов заданий с комментариями и координатами. Они добавляют переданные параметры в данные результата.

```

inline void ResultB(const char* Cmt, bool B, int X, int Y);
inline void ResultN(const char* Cmt, int N, int X, int Y, int W);
inline void ResultN(const char* Cmt, int N1, int N2, int X, int Y, int W);
inline void ResultN(const char* Cmt, int N1, int N2, int N3, int X, int Y,
                    int W);
inline void ResultR(const char* Cmt, double R, int X, int Y, int W);
inline void ResultR(const char* Cmt, double R1, double R2, int X,
                    int Y, int W);
inline void ResultR(const char* Cmt, double R1, double R2, double R3,
                    int X, int Y, int W);
inline void ResultC(const char* Cmt, char C, int X, int Y);
inline void ResultS(const char* Cmt, const char* S, int X, int Y);
inline void ResultComment(const char* Cmt, int X, int Y);

```

Эти методы добавляют файлы к исходным данным, передавая имя файла и аргументы соответствующей внутренней функции.

```

inline void DataFileN(const char* FileName, int Y, int W);
inline void DataFileR(const char* FileName, int Y, int W);
inline void DataFileC(const char* FileName, int Y, int W);
inline void DataFileS(const char* FileName, int Y, int W);
inline void DataFileT(const char* FileName, int Y1, int Y2);

```

Аналогично, эти методы добавляют файлы к данным результата, передавая имя файла и аргументы соответствующей внутренней функции.

```

inline void ResultFileN(const char* FileName, int Y, int W);
inline void ResultFileR(const char* FileName, int Y, int W);
inline void ResultFileC(const char* FileName, int Y, int W);
inline void ResultFileS(const char* FileName, int Y, int W);
inline void ResultFileT(const char* FileName, int Y1, int Y2);

```

Эти методы используются для получения различных статистических данных и образцов текста. Они возвращают количество слов, предложений и абзацев соответственно, а также образец текста по указанному индексу.

```

inline int EnWordCount();
inline int EnSentenceCount();
inline int EnTextCount();
inline std::string EnWordSample(int N);
inline std::string EnSentenceSample(int N);
inline std::string EnTextSample(int N);

```

Метод `activate` используется для загрузки и активации указанного разделяемого объекта (разделяемой библиотеки).

```

inline void activate(const char * DllName);

```

`SetProcess` реализует функцию для установки рейтинга текущего процесса.

```

inline void SetProcess(int ProcessRank);

```

`CurrentTest` используется для получения номера текущего теста. `CurrentLanguage` и `CurrentVersion` используются для получения языка и версии текущей задачи соответственно.

```

inline int CurrentTest();
inline int CurrentLanguage();
inline char* CurrentVersion();

```

`SetGroupData` устанавливает основные параметры группы, включая название, описание, автора, ключевые слова и количество задач.

```

inline void SetGroupData(const char* topic, const char*
descr, const char* author, const char* key, int n);

```

`ClearGroupData` используется для очистки всех данных для текущей группы. Эта функция не имеет параметров и выполняет единственную функцию - очистку данных группы.

```

inline void ClearGroupData();

```


GetTopic и GetTaskCount используются для получения темы текущей группы и количества задач в текущей группе, соответственно. Аналогично, GetTestCount и GetCode используются для получения количества тестов и текущей строки кода, соответственно.

```
inline std::string GetTopic();  
inline int GetTaskCount();  
inline int GetTestCount();  
inline std::string GetCode();
```

SetTaskData передает номер задачи и номер теста для установки данных задачи. SetLanguage используется для установки текущего языка (английский или русский). SetFileRow передает целое число N для установки номера текущей строки файла. Эта функция используется для поиска определенной строки в файле.

```
inline std::string SetTaskData(int task, int test);  
inline void SetLanguage(int n);  
inline void SetFileRow(int N);
```

Pause используется для приостановки текущей операции.

```
inline void Pause();
```

NoEraseNextFile задает следующий файл, который не будет стираться. Эта функция может использоваться для предотвращения случайного удаления файлов при их обработке.

```
inline void NoEraseNextFile();
```

RandomN и RandomR используются для ввода диапазонов и генерации случайных чисел (целых и плавающих).

```
inline int RandomN(int M, int N);  
inline double RandomR(double A, double B);
```

3 Перенос компонентов PT5 и конструктора в Linux

В современных операционных системах Dynamic Link Library (DLL) и Shared Object (SO) - две широко используемые технологии для модульности и повторного использования кода. Хотя концептуально они похожи, существуют значительные различия в том, как они реализуются и используются в разных операционных системах. Это основная проблема, с которой мы столкнулись при переносе компонентов и конструкторов PT5 в Linux.

3.1 Особенности переноса в Linux

В операционной системе Windows библиотека динамических связей реализована в формате PE (Portable Executable) с расширением файла .dll. При вызове DLL Windows использует функцию LoadLibrary для ее динамической загрузки и получает адрес экспортируемой функции с помощью функции GetProcAddress. Функции можно экспортировать, используя declspec(dllexport) в объявлении функции или явно перечисляя экспортируемые функции в файле определения модуля (.def). Этот механизм гарантирует, что определенные функции в DLL могут быть вызваны другими приложениями или DLL. Кроме того, Windows поддерживает использование таких соглашений о вызове, как stdcall и cdecl, для обеспечения согласованности и стандартизации вызовов функций. Стоит отметить, что когда разные процессы загружают одну и ту же DLL, глобальные переменные каждого процесса независимы и не влияют друг на друга.

Операционная система Linux, напротив, использует общие объекты с расширением файла .so и формат ELF (Executable and Linkable Format). Linux динамически загружает общие библиотеки с помощью функции dlopen и использует функцию dlsym для получения адресов экспортируемых функций. В отличие от Windows, Linux экспортирует все нестатические функции по умолчанию, но это можно контролировать с помощью опций компоновщика. Такое поведение по умолчанию упрощает использование общих библиотек, но может привести к ненужному раскрытию символов. Что касается соглашений о вызове функций, Linux обычно использует стандартные соглашения о вызове C без необходимости объявления специальных соглашений о вызове. Как и в Windows, глобальные переменные общих библиотек независимы, когда разные процессы загружают одну и ту же общую библиотеку. Столбцы префикса и

расширения библиотеки указывают, как разрешаются и создаются имена библиотек для перечисленных операционных систем.

Operating System	Dynamic library extension	Static library extension	Library prefix
FreeBSD	.so	.a	lib
macOS	.dylib	.a	lib
Linux	.so	.a	lib
Windows	.dll	.lib	n/a
Haiku	.so	.a	lib

Рис. 1: Библиотеки для различных операционных систем

Работа с загрузкой и использованием динамических библиотек в среде Lazarus требует подхода, подходящего для каждой операционной системы. Загрузка динамических библиотек, получение адресов функций и выгрузка библиотек различны для операционных систем Windows и Linux. Поэтому мы также изменили соответствующие функции.

Использование библиотек динамических связей (DLL) в Windows включает в себя три основные функции. Во-первых, функция `LoadLibrary` используется для загрузки DLL-файла. Например, вы можете использовать `LibHandle := LoadLibrary('example.dll');` для загрузки библиотеки динамических ссылок с именем `example.dll`. После успешной загрузки возвращается библиотечный хэндл `LibHandle` (типа `THandle`). Далее используйте функцию `GetProcAddress`, чтобы получить адрес функции в DLL. В частности, адрес функции `ExampleFunction` можно получить с помощью функции `@ExampleFunction := GetProcAddress(LibHandle, 'ExampleFunction');`. Если это удалось, можно вызвать `ExampleFunction`. После использования DLL необходимо вызвать функцию `FreeLibrary` для удаления DLL, чтобы убедиться, что ресурсы освобождены, код - `FreeLibrary(LibHandle);`.

Для использования общих библиотек (файлов `.so`) в системах Linux также необходимы три основные функции. Во-первых, функция `LoadLibrary` используется для загрузки файла общей библиотеки, например, `LibHandle := LoadLibrary('example.so');`. После успешной загрузки возвращается библиотечный хэндл `LibHandle` (типа `TLibHandle`). Затем с помощью функции `GetProcedureAddress` получите адрес функции в общей библиотеке, например, `ExampleFunction := GetProcedureAddress(LibHandle, 'ExampleFunction');`. После успешного получения адреса можно вызывать функцию. После завершения использования необходимо вы-

звать функцию `UnloadLibrary`, чтобы выгрузить разделяемую библиотеку, код - `UnloadLibrary(LibHandle);`.

```
procedure FreeLib(var handle: TLibHandle);
// Разрушение дескриптора динамической библиотеки
begin
  if handle <> 0 then
  begin
    {$IFDEF WINDOWS}
      FreeLibrary(handle);
    {$ENDIF}
    {$IFDEF LINUX}
      UnloadLibrary(handle);
    {$ENDIF}
    handle := 0;
  end;
end;
```

Кроме того, в процессе переноса компонентов и конструкторов PT5 на Linux мы изменили путь к компилятору, который хранится в файле `pt5comprmpi` в папке `header` (папка C или CPP). Ниже приведены файлы `pt5comprmpi` для различных сред Linux и Windows:

```
addfiles=pt5mpi.cpp;pt5mpi.hpp;.vscode\c_cpp_properties.json;
.vscode\launch.json;.vscode\tasks.json;mpi.h;mssmpi.lib
compiler=C:\Program Files (x86)\Embarcadero\Dev-Cpp\TDM-GCC-64\bin\g++.exe
cargs=-fdiagnostics-color=always||-Wall||-m32||-std=c++14||${fileDirname}\
${fileName}||${fileDirname}\pt5mpi.cpp||-o||${fileDirname}\ptprj.exe||${file
Dirname}\mssmpi.lib
cwd=C:\Program Files (x86)\Embarcadero\Dev-Cpp\TDM-GCC-64\bin
exename=${fileDirname}\ptprj.exe
run=c:\Program Files\Microsoft MPI\bin\mpiexec.exe
rargs=-n||${procNum}||${exeName}
rwd=${fileDirname}
timeout=5000
```

Пути к компиляторам и тип компилятора, используемого в операционных системах Windows и Linux, существенно различаются. Путь `C:\Program Files`

(x86)\Embarcadero\Dev-Cpp\TDM-GCC-64\bin\g++.exe представляет собой компилятор G++, используемый в среде Windows. Компилятор является частью TDM-GCC и обычно ассоциируется с интегрированной средой разработки (IDE) Dev-C++ компании Embarcadero. Windows использует обратную косую черту (\) в качестве разделителя путей, и путь компилятора указывает, что он установлен в определенном каталоге системы. Компилятор G++ в основном используется для компиляции стандартного кода C++, генерируя исполняемые файлы (.exe) для платформы Windows.

В противоположность этому, путь /usr/bin/mpicxx представляет компилятор MPICXX, используемый в среде Linux, который обычно является оберткой компилятора C++ для интерфейса передачи сообщений (MPI) для параллельных вычислений. Компилятор MPICXX используется для компиляции параллельного кода на C++, требующего поддержки MPI, для создания исполняемых файлов на платформе Linux. Ниже приведен файл pt5complmpi.cpp, модифицированный для среды Linux:

```
addfiles=pt5mpi.cpp;pt5mpi.hpp;.vscode/c_cpp_properties.json;
    .vscode/launch.json;.vscode/tasks.json
compiler=/usr/bin/mpicxx
cargs=-fdiagnostics-color=always||-Wall||-std=c++14||
    ${fileDirname}/${fileName}||${fileDirname}/pt5mpi.cpp||-o||
    ${fileDirname}/ptprj
cwd=/usr/bin/
exename=${fileDirname}/ptprj
run=/usr/bin/mpiexec
rargs=-n||${procNum}||${exeName}
rwd=${fileDirname}
timeout=5000
```

Кроме того, мы несколько изменили параметры компиляции (cargs). Параметр компиляции -m32 используется для указания компилятору генерировать 32-битный объектный код. В среде Windows параметр -m32 используется в основном для того, чтобы обеспечить генерацию 32-битного объектного кода для совместимости со старыми системами и 32-битными приложениями и библиотеками. Хотя современные операционные системы Windows (такие как Windows 10 и Windows 11) в основном 64-битные, многие приложения Windows должны быть совместимы с 32-битными системами и старыми аппаратными архитектурами. Некоторые среды разработки и цепочки инструментов по умолчанию генерируют 32-битные исполняемые файлы,

чтобы обеспечить широкую совместимость. В частности, при использовании таких компиляторов, как TDM-GCC, разработчикам может потребоваться явно указать генерацию 32-битного кода, чтобы удовлетворить потребности старых систем и программных сред. Кроме того, в некоторых проектах могут быть особые требования к генерации 32-битных приложений для конкретной среды выполнения или для совместимости с определенными библиотеками и зависимостями. По историческим причинам многие приложения для Windows были разработаны и развернуты на 32-битных системах, и для продолжения поддержки этих приложений разработчикам может потребоваться создание 32-битных версий.

В отличие от этого, в среде Linux параметр `-m32` обычно не используется, поскольку большинство современных дистрибутивов Linux по умолчанию устанавливаются и работают на 64-битных архитектурах. 64-битные системы обеспечивают лучшую производительность, большее адресное пространство и большую безопасность, и поэтому становятся все более распространенными. Компилятор GCC в Linux по умолчанию генерирует 64-битный объектный код, если только параметр `-m32` не указан явно для генерации 32-битный код. Поскольку большинство библиотек с открытым исходным кодом и зависимостей уже доступны в 64-битных версиях, разработчикам обычно не нужно генерировать 32-битный код для удовлетворения зависимостей. В среде Linux большинство современных приложений и системных компонентов перешли на 64-битную архитектуру, поэтому потребность в 32-битных приложениях снижается. Разработчики в Linux часто настраивают и оптимизируют код для 64-битных сред, чтобы воспользоваться преимуществами производительности современных процессоров.

3.2 Проблемы, возникшие при переносе

Одной из значимых проблем, возникших при переносе программного обеспечения с Windows на Unix-системы, является различие в учёте регистра символов в именах файлов. В Windows имена файлов не чувствительны к регистру, что означает, что файлы `"example.dll"` и `"Example.DLL"` будут считаться одним и тем же файлом. Однако в Unix-системах, таких как Linux, регистр символов в именах файлов учитывается, и файлы `"example.so"` и `"Example.so"` будут рассматриваться как два различных файла.

Эта проблема создает особые трудности при использовании динамических библиотек. Например, если попытаться найти и загрузить библиотечный файл, а его имя

не соответствует регистру, система Unix не сможет найти нужный файл, в то время как в Windows такой проблемы нет. Хорошим решением будет преобразовать все имена библиотечных файлов в нижний регистр, чтобы избежать подобных проблем. Обратите внимание, что это относится не только к именам библиотечных файлов, но и к другим операциям файловой системы.

Существует также проблема, связанная с использованием имен программ. Динамические библиотеки обычно возвращают «правильное» имя группы задач, которое содержит правильную комбинацию регистров. Поэтому в Unix-системах необходимо разработать механизм проверки и исправления имен файлов. Если в команде `pt5run` указано имя «`democrp1`», а файл не существует, ищите файл с правильным регистром, например «`DemoCpP1`». Если файл с правильным именем существует, его следует использовать; если он также не существует, следует создать файл с правильным именем. Если файл «`democrp1`» существует, необходимо обработать только «`democrp1`» и не создавать новый файл.

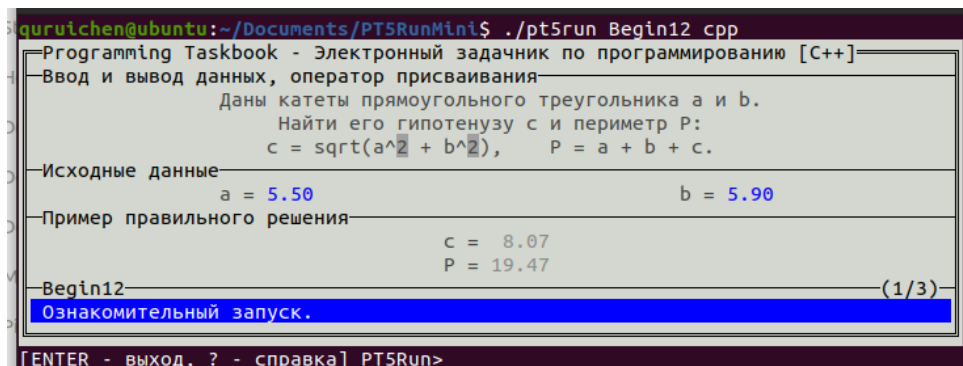
Этот дополнительный механизм проверки и коррекции имен файлов должен применяться, прежде всего, в версиях программы для ОС Linux, и поможет избежать многих проблем, связанных с чувствительностью к регистру. Таким образом, правильное управление именами файлов и учет особенностей операционной системы могут значительно упростить процесс переноса программного обеспечения и повысить его надежность и совместимость.

4 Примеры работы задачника и конструктора для Linux

Решение задачи должно быть сохранено в файле с тем же именем, что и название задачи, тип файла зависит от команды. Чтобы запустить PT5 для проверки правильности решения, введите в терминале команду `./pt5run`, имя задачи, которую вы хотите просмотреть или проверить и краткое название языка программирования. Например, чтобы просмотреть вторую задачу на языке cpp в группе Begin, просто введите `./pt5run Begin2 cpp`.

4.1 Примеры работы задачника для Linux

В следующем примере показано, как запустить задание в демонстрационном режиме с помощью команды `pt5run`:



```
guruichen@ubuntu:~/Documents/PT5RunMini$ ./pt5run Begin12 cpp
Programming Taskbook - Электронный задачник по программированию [C++]
Ввод и вывод данных, оператор присваивания
Даны катеты прямоугольного треугольника a и b.
Найти его гипотенузу c и периметр P:
c = sqrt(a^2 + b^2), P = a + b + c.
Исходные данные
a = 5.50 b = 5.90
Пример правильного решения
c = 8.07
P = 19.47
Begin12 (1/3)
Ознакомительный запуск.
[ENTER - выход, ? - справка] PT5Run>
```

Рис. 2: Задача Begin12 в cpp

Другие команды, доступные в стандартном режиме, можно найти, набрав `?` чтобы запросить больше команд, которые можно использовать в стандартном режиме. Ниже показаны все команды, доступные в стандартном режиме:


```

guruichen@ubuntu:~/Documents/PT5RunMini$ ./pt5run if5 c
Programming Taskbook [C]
-Conditional statement-
    Three integers are given. Find the amount of positive
    and amount of negative integers in the input data.
-Input data-
    0          0          -17
-Example of right solution-
    Amount of positive integers:  0
    Amount of negative integers:  1
-If5- (1/9)
Acquaintance with the task.

[ENTER - quit, ? - info] PT5Run> ?
Programming Taskbook [C]
-Conditional statement-
    Three integers are given. Find the amount of positive
    and amount of negative integers in the input data.
-Input data-
    0          0          -17
-Example of right solution-
    Amount of positive integers:  0
    Amount of negative integers:  1
-If5- (1/9)
Acquaintance with the task.

=== Programming Taskbook 5.1 for C (standard mode) ===
? - show this info,      f - collapse/expand content of text files,
a - hide/show additional info if additional info is not empty,
s - swap task text section and data sections,
t - hide/show task text, d - hide/show task info if debug info is not empty,
# - show task in html format,
w - white color scheme, b - black color scheme, n - no color, ENTER - quit
PT5Run>

```

Рис. 3: Команды в стандартном режиме

```

guruichen@ubuntu:~/Documents/PT5RunMini$ ./pt5run File25_en cpp
Programming Taskbook [C++]
-Binary operations-
    Given a file of real numbers,
    replace the values of all file components with their squares.
-Input data-
    File name: "g7k40bm5.tst"          File contents:
    1:  1.50  7.80  4.50  5.30  8.80  6.80  9.10  5.30  7.20  7.10
    11: 4.60  8.80  7.20  5.30  2.80  4.70  1.90  9.90  8.00
-Output data-
    New contents of the file:
    1:  2.25 60.84 20.25 28.09 77.44 46.24 82.81 28.09 51.84 50.41
    11: 21.16 77.44 51.84 28.09 7.84 22.09 3.61 98.01 64.00
-File25- (5/5)
The task is solved!

[ENTER - quit, ? - info] PT5Run>

```

Рис. 4: Пример правильного решения задачи File25

```
guruilchen@ubuntu:~/Documents/PT5RunMini$ ./pt5run String41 c
Programming Taskbook [C]
Characters and strings: word processing
A string that contains English words separated by one
or more blank characters is given. Find the amount of words in the string.
Input data
"MAGNIFICATION SINGING SLEEVE SINGING MANUFACTURER"
Output data
4
Example of right solution
5
String41 (1/5)
Wrong solution.
[ENTER - quit, ? - info] PT5Run>
```

Рис. 5: Пример неправильного решения задачи String41

```
guruilchen@ubuntu:~/Documents/PT5RunMini$ ./pt5run If5 cpp
Programming Taskbook [C++]
Conditional statement
Three integers are given. Find the amount of positive
and amount of negative integers in the input data.
Input data
0 0 -17
Output data
Amount of positive integers: 0
Amount of negative integers: 0
Example of right solution
Amount of positive integers: 0
Amount of negative integers: 1
If5 (1/9)
An attempt to output superfluous data.
[ENTER - quit, ? - info] PT5Run>
```

Рис. 6: Попытка ввода избыточных данных для задачи If1

Порождающие паттерны

Factory Method, Abstract Factory

Product

ConcreteProduct

Creator

ConcreteCreator

FactoryMethod

FactoryMethod

product = FactoryMethod()

return new ConcreteProduct

Factory Method (Фабричный метод) — порождающий паттерн.

Известен также под именем **Virtual Constructor (Виртуальный конструктор)**.

Частота использования: высокая.

Назначение: определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать. Фабричный метод позволяет классу делегировать инстанцирование подклассам.

Участники:

- **Product (Продукт)** — определяет интерфейс объектов, создаваемых фабричным методом;
- **ConcreteProduct (Конкретный продукт)** — реализует интерфейс Product;
- **Creator (Создатель)** — объявляет фабричный метод, возвращающий объект типа Product; может также определять реализацию фабричного метода по умолчанию, возвращающую некоторый конкретный продукт; реализует методы, в которых используется объект Product, созданный фабричным методом;
- **ConcreteCreator (Конкретный создатель)** — замещает фабричный метод для создания конкретного продукта.

Задавание 1. Реализовать две иерархии классов, в одну из которых входит абстрактный создатель Creator и два конкретных создателя ConcreteCreator1 и ConcreteCreator2, а в другую — абстрактный продукт Product и два конкретных продукта ConcreteProduct1 и ConcreteProduct2.

Абстрактный класс Product содержит два абстрактных метода, связанных с получением и преобразованием строки: метод getInfo без параметров, возвращающий строку, и метод Transform без параметров, который ничего не возвращает. Классы ConcreteProduct1 и ConcreteProduct2 содержат строковое поле info, которое инициализируется в конструкторе с помощью одноименного параметра, после чего в конструкторе класса ConcreteProduct1 поле info преобразуется к шапке регистра, а в конструкторе класса ConcreteProduct2 — к верхнему. Метод getInfo в каждом подклассе возвращает текущее значение поля info, а метод Transform преобразует это поле следующим образом: для ConcreteProduct1 он добавляет дополнительный пробел после каждого непробельного символа поля info (кроме его последнего символа), а для ConcreteProduct2 он добавляет два дополнительных символа * (звездочка) после каждого символа, отличного от звездочки (кроме последнего символа).

Абстрактный класс Creator содержит абстрактный фабричный метод FactoryMethod(info) со строковым параметром info, возвращающий ссылку на объект Product. Этот метод определяется в классах ConcreteCreator1 и ConcreteCreator2. Прием фабричный метод класса ConcreteCreator1 создает объект типа ConcreteProduct1, а фабричный метод класса ConcreteCreator2 создает объект типа ConcreteProduct2; в любом случае конструктору создаваемого объекта передается параметр info фабричного метода.

Рис. 7: Образец HTML-документа с формулировкой задачи OOP1Creat1

4.2 Демонстрационная группа для конструктора C++

Демонстрационная группа DemoCpp показывает, как использовать PT5TaskMaker для создания новых заданий с применением конструктора для языка c++. В отличие от большинства других групп, подготовленных с применением конструктора для языка Free Pascal и откомпилированных в среде Lazarus, группа DemoCpp представляет собой объектный файл (файл.so), созданный конструктором C++ и скомпилированный CMake в среде Linux. Ниже приведен Файл CMakeLists.txt, используемый для создания файла pt5democpp.so, показывающий, как настроить проект, добавить исходные файлы и сгенерировать файлы общих объектов (файлы .so)

```
1 project(stllib)
2 cmake_minimum_required(VERSION 3.0)
3
4 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++17")
5 SET(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR}/bin)
6 SET(LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib)
7
8 aux_source_directory(./src SRC)
9
10 include_directories(./inc)
11
12
13
14 add_compile_options(-std=c++17)
15
16 add_library("pt5democpp" SHARED ${SRC})
```

Рис. 8: пример файла CMakeLists.txt

Также приведен пример исходного файла democpp.cpp, демонстрирующего простую задачу, которая будет компилироваться в общий объектный файл:

```

void MakerDemo4()
{
    CreateTask("Двумерные массивы (матрицы): вывод элементов");
    TaskText("Дана матрица размера-{M}\\x\\x{N} и целое число-{K} (1-\\l-{K}-\\l-{M}).", 0, 2);
    TaskText("Вывести элементы {K}-й строки данной матрицы.", 0, 4);
    int m = RandomN(2, 5), n = RandomN(4, 8), k = 1;
    if (m == 5)
    {
        k = 0;
        DataN("M = ", m, 3, 1, 1);
        DataN("N = ", n, 10, 1, 1);
        double a[5][8];
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
            {
                a[i][j] = RandomR(-9.99, 9.99);
                DataR(a[i][j], Center(j+1, n, 5, 1), i + k + 1, 5);
            }
        k = RandomN(1, m);
        DataN("K = ", k, 68, 5, 1);
        for (int j = 0; j < n; j++)
            ResultR(a[k-1][j], Center(j+1, n, 5, 1), 3, 5);
        SetTestCount(5);
    }
}

```

Рис. 9: пример файла ДемоСрр.срр

Запустить демонстрационную группу (группа ДемоСрр) можно через терминал, переместив соответствующий сгенерированный файл общих объектов (файл.so) в папку lib. Ниже приведен пример отображения заданий из группы ДемоСрр.

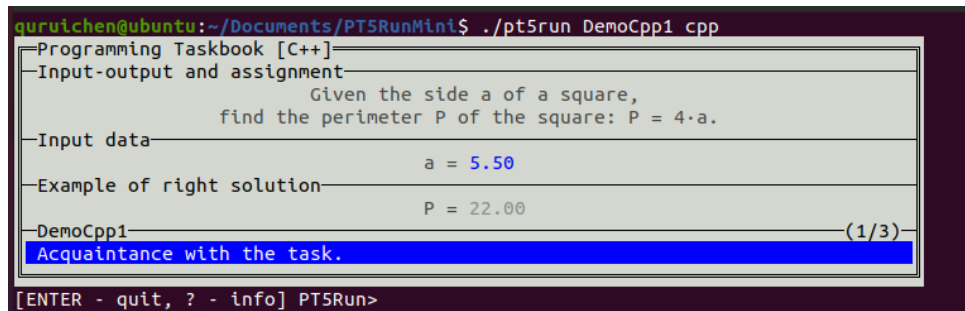


Рис. 10: ДемоСрр1

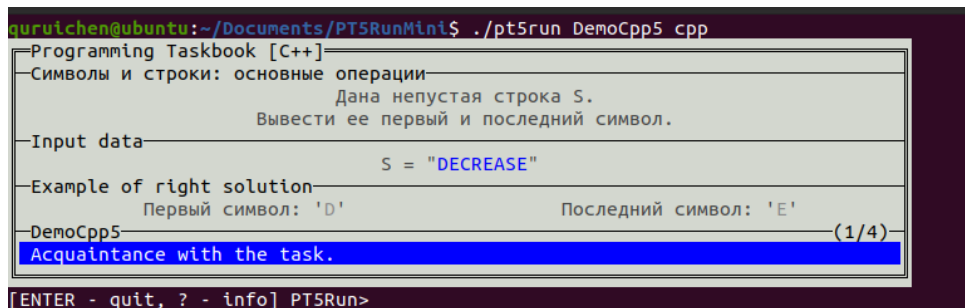


Рис. 11: ДемоСрр5

После генерации новой группы и ее размещения в нужной папке все входящие в нее задания можно выполнять на любом языке, поддерживаемом задачиком, ис-

пользуя средства для ввода исходных данных и вывода результатов и отладочных данных, предусмотренные в задачнике для этого языка. Ниже приведены примеры решений задач из группы DemoC++ для языков C и C++.

```

guruichen@ubuntu:~/Documents/PT5RunMini$ ./pt5run DemoC++3 cpp
Programming Taskbook [C++]
Ввод и вывод данных, оператор присваивания
Даны стороны прямоугольника a и b.
Найти его площадь S = a·b и периметр P = 2·(a + b).
Input data
a = 1.80 b = 1.40
Example of right solution
S = 2.52
P = 6.40
DemoC++3 (1/3)
Acquaintance with the task.
[ENTER - quit, ? - info] PT5Run>

```

Рис. 12: DemoC++3

```

DemoC++3.cpp
DemoC++3.cpp > ...
1  #include "pt5.hpp"
2
3  using namespace std;
4  void Solve()
5  {
6      Task("DemoC++3");
7  }
8
9
10

```

Рис. 13: DemoC++3

Пример правильного решения:

```
DemoCpp3.cpp X
DemoCpp3.cpp > ...
1  #include "pt5.hpp"
2
3  using namespace std;
4  void Solve()
5  {
6      Task("DemoCpp3");
7      double a,b;
8      pt>>a>>b;
9      pt<<(a*b)<<2*(a+b);
10
11 }
12
13
```

Рис. 14: DemoCpp3

```
guruichen@ubuntu:~/Documents/PT5RunMini$ ./pt5run DemoCpp3 cpp
Programming Taskbook [C++]
Ввод и вывод данных, оператор присваивания
Даны стороны прямоугольника a и b.
Найти его площадь S = a·b и периметр P = 2·(a + b).
Input data
a = 7.80 b = 9.90
Output data
S = 77.22
P = 35.40
DemoCpp3 (3/3)
The task is solved!
[ENTER - quit, ? - info] PT5Run>
```

Рис. 15: DemoCpp3

Ниже приведены примеры решения проблем, предложенные группой String для языка C.

```
guruichen@ubuntu:~/Documents/PT5RunMini$ ./pt5run String41_en c
Programming Taskbook [C]
Characters and strings: word processing
  A string that contains English words separated by one
  or more blank characters is given. Find the amount of words in the string.
Input data
  "MAGNIFICATION   SINGING   SLEEVE   SINGING   MANUFACTURER"
Example of right solution
  5
String41 (1/5)
Acquaintance with the task.
```

Рис. 16: String41

Исходный файл String41.c выглядит следующим образом.

```
C String41.c X
C String41.c
1  #include "pt5.h"
2
3  void Solve()
4  {
5      Task("String41");
6  }
7
8
9
```

Рис. 17: String41

Правильное решение заключается в следующем.

```

C String41.c
C String41.c > ...
1  #include "pt5.h"
2  #include <ctype.h>
3
4  int countWords(char *str)
5  {
6      int count = 0;
7      int inWord = 0;
8
9      while (*str) {
10         if (isspace(*str)) {
11             inWord = 0;
12         } else {
13             if (!inWord) {
14                 count++;
15                 inWord = 1;
16             }
17         }
18         str++;
19     }
20     return count;
21 }
22 void Solve()
23 {
24     Task("String41");
25     char str[999];
26     GetS(str);
27     int wordCount = countWords(str);
28     PutN(wordCount);
29 }
30

```

Рис. 18: String41

Запустив его в терминале, вы увидите следующее.

```

guruichen@ubuntu:~/Documents/PT5RunMini$ ./pt5run String41 en c
Programming Taskbook [C]
Characters and strings: word processing
A string that contains English words separated by one
or more blank characters is given. Find the amount of words in the string.
Input data
REPRODUCER INVOCATION
Output data
2
String41 (5/5)
The task is solved!
[ENTER - quit, ? - info] PT5Run>

```

Рис. 19: String41

4.3 Реализация для задачника РТ5 групп заданий по изучению библиотеки STL

Наличие конструктора для языка с++ позволило легко адаптировать к версии 5 задачника те группы заданий, которые были ранее разработаны для версии 4. При этом адаптация была выполнена как для варианта задачника РТ5 для ОС Windows, так и для варианта для ОС Linux.

В задачник РТ5 были перенесены все группы заданий, входящие в расширение задачника для изучения стандартной библиотеки шаблонов С++, а именно, следующие группы:

Библиотека STL состоит из шести основных групп - Iter, Seq, Alg, Str, Assoc и Func. Она компилируется с помощью комбинации libpt5stl_ru.cpp (русский) и libpt5_en.cpp (английский). Эти группы скомпилированы в специальную библиотеку для pt5run, из которой он берет задачи, связанные с STL.



Рис. 20: STL

Ниже приводится пример формирования и отображения на экране одной из задач группы STL1Iter. Далее приводится пример правильного решения этой задачи.

```
void Stl6()
{
    CreateTask();
    TaskText(
        "Дана строка {name} и набор символов. Записать в текстовый файл\n"
        "с именем {name} исходный набор символов в том же порядке, добавляя\n"
        "после каждого символа пробел. Использовать итераторы ptin\\_iterator,\n"
        "ostream\\_iterator и алгоритм copy."
    );
    string name = RandomStr(8) + ".tst";
    DataS("name = ", name.c_str(), 0, 2);
    MakeVC(RandomN(10, 34), abext);
    DataVC("", 3);
    // DataFileVN(name.c_str(), 2, 5);
    ResultFileVC(name.c_str(), 3, 3, " ");
}
```

```

guruichen@ubuntu:~/Documents/PT5RunMini$ ./pt5run STL1Iter6 cpp
Programming Taskbook [C++]
Знакомство с итераторами и алгоритмами
Дана строка name и набор символов. Записать в текстовый файл
с именем name исходный набор символов в том же порядке, добавляя
после каждого символа пробел. Использовать итераторы ptin_iterator,
ostream_iterator и алгоритм copy.
Input data
name = "kuqzlxzo.tst"
(22): 'A' '4' 'Z' 'P' 'R' 'V' '2' 'c' 'o' '6' 'e' 'w' 'm' 'K' 'X' 'q' 'C'
      'b' 'o' 'Y' 'q' 'x'
Example of right solution
1: "A 4 Z P R V 2 c o 6 e w m K X q C b o Y q x "
STL1Iter6 (1/5)
Acquaintance with the task.
[ENTER - quit, ? - info] PT5Run>

```

Рис. 21: STL1Iter6

Решение следующее:

```

#include "pt5.hpp"
#include<iostream>
#include<vector>
#include<algorithm>
#include <fstream>
#include<iterator>
using namespace std;
typedef ptin_iterator<char> ptin;
void Solve()
{
    Task("STL1Iter6");
    string s;
    pt>>s;
    ofstream os(s);
    copy(ptin(0), ptin(),ostream_iterator<char>(os, " "));
    os.close();
}

```

Проверить ответ можно, запустив в терминале `./pt5run STL1Iter6 cpp`, чтобы проверить правильность ответа.

```

guruichen@ubuntu:~/Documents/PT5RunMini$ ./pt5run STL1Iter6 cpp
Programming Taskbook [C++]
Знакомство с итераторами и алгоритмами
Дана строка name и набор символов. Записать в текстовый файл
с именем name исходный набор символов в том же порядке, добавляя
после каждого символа пробел. Использовать итераторы ptin_iterator,
ostream_iterator и алгоритм copy.
Input data
name = "kp0q4at3.tst"
(19): 'T' 'T' 't' 'f' 'i' 'p' 'Z' 'S' 'E' '4' 'k' 'R' 'j' 'e' 'e' 'B' 'n'
      'L' 'B'
Output data
1: "T T t f i p Z S E 4 k R j e e B n L B "
STL1Iter6 (5/5)
The task is solved!
[ENTER - quit, ? - info] PT5Run>

```

Рис. 22: STL1Iter6

4.4 Реализация для задачника РТ5 группы заданий по изучению параллельных матричных алгоритмов

В задачник РТ5 была перенесена завершающая группа MPI9Matr из расширения задачника, посвященного параллельному программированию на базе MPI. В этой группе рассматриваются различные варианты алгоритма параллельного умножения матриц.

Ниже приводится пример формирования и отображения на экране одной из задач данной группы. Далее приводится пример правильного решения этой задачи.

```

void Matr1ScatterFile() {
n = RandomN(3, 5);
CreateTask(topic1, &n);
if (n == 0)
return;
TaskText(
"Integers {M}, {P}, {Q} and two file names are given in the master\n"
"process. The given files contain elements of a matrix {A} of the\n"
"size {M}~\\x~{P} and a matrix {B} of the size {P}~\\x~{Q}. Modify the\n"
"initial stage of the band algorithm of matrix multiplication\n"
"(see the \\6 task) as follows: each process should read the\n"
"corresponding bands of the matrices {A} and {B} directly from\n"
"the given files using the MPI\\_File\\_seek and MPI\\_File\\_read\\_all\n"
"collective functions (a new file view is not required).\n"
"\\PTo send the sizes of matrices and file names, use the\n"

```

```

"MPI\\_Bcast collective function.\n"
"\\P\\Include all these actions in a Matr1ScatterFile function\n"
"(without parameters). As a result of the call of this function,\n"
"each process will receive the values {N}_{A}, {P}, {N}_{B}, {Q},\n"
"as well as one-dimensional arrays filled with the corresponding\n"
"bands of the matrices {A}, {B}, {C}. Output all obtained data\n"
"(that is, the numbers {N}_{A}, {P}, {N}_{B}, {Q} and the bands of\n"
"the matrices {A}, {B}, {C}) in each process after calling the\n"
"Matr1ScatterFile function. Perform the input of initial data in\n"
"the Matr1ScatterFile function, perform the output of the results\n"
"in the Solve function.\n"
"\\P\\SRemark.\\s For some bands, some of their elements (namely,\n"
"the last rows) or even the entire bands should not be read from\n"
"the source files and will remain zero-valued ones. However, this\n"
"situation does not require special processing, since the\n"
"MPI\\_File\\_read\\_all function automatically stops reading the\n"
"data (without generating any error message) when the end of the\n"
"file is reached.");
MakeDim1();
Init1();
y = 1;
DataProcComment(0, y);
DataN("M = ", m, 12, y, 1);
DataN("P = ", p, 20, y, 1);
DataN("Q = ", q, 28, y++, 1);
std::string sa = "A" + FileName(5) + ".tst",
            sb = "B" + FileName(5) + ".tst";
std::fstream fa(sa.c_str(), std::ios_base::binary |
                std::ios_base::out),
            fb(sb.c_str(), std::ios::binary |
                std::ios_base::out);
for (int i = 0; i < m*p; i++)
    fa.write((char*)&a[i], sizeof(a[i]));
for (int i = 0; i < p*q; i++)
    fb.write((char*)&b[i], sizeof(a[i]));

```

```

fa.close();
fb.close();
DataS("File with matrix A: ", sa.c_str(), 1, y++);
SetFileRow(p);
DataFileN(sa.c_str(), y++, 4);
DataS("File with matrix B: ", sb.c_str(), 1, y++);
SetFileRow(q);
DataFileN(sb.c_str(), y++, 4);
y = 1;
for (int i = 0; i < n; i++) {
    ResultProcComment(i, y);
    ResultN("N_A = ", k1, 12, y, 1);
    ResultN("P = ", p, 21, y, 1);
    ResultN("N_B = ", k2, 29, y, 1);
    ResultN("Q = ", q, 38, y++, 1);
    ResultMatr1(ind("A", i, ":").c_str(), &a[i*k1*p], k1, p, p, y);
    ResultMatr1(ind("B", i, ":").c_str(), &b[i*k2*q], k2, q, q, y);
    ResultMatr1(ind("C", i, ":").c_str(), &c[i*k1*q], k1, q, q, y);
}
Delete1();
}

```

```

qurui@ubuntu:~/Documents/PT5RunMini$ ./pt5run MPI9Matr8 cpp
"/usr/bin/mpiexec" -n 3 /home/qurui@ubuntu/Documents/PT5RunMini/ptprj
Programming Taskbook [C++]
Band algorithm 1 (horizontal bands)
  Integers M, P, Q and two file names are given in the master process.
  The given files contain elements of a matrix A of the size M × P
  and a matrix B of the size P × Q. Modify the initial stage of the band
  algorithm of matrix multiplication (see the MPI9Matr2 task) as follows:
  each process should read the corresponding bands of the matrices A and B
  directly from the given files using the MPI_File_seek and MPI_File_read_all
  collective functions (a new file view is not required).
  To send the sizes of matrices and file names, use the MPI_Bcast
  collective function.
  Include all these actions in a MatrisScatterFile function (without parameters).
  As a result of the call of this function, each process will receive
  the values NA, P, NB, Q, as well as one-dimensional arrays filled
  with the corresponding bands of the matrices A, B, C. Output all obtained data
  (that is, the numbers NA, P, NB, Q and the bands of the matrices A, B, C)
  in each process after calling the MatrisScatterFile function. Perform the input
  of initial data in the MatrisScatterFile function, perform the output
  of the results in the Solve function.
  Remark. For some bands, some of their elements (namely, the last rows)
  or even the entire bands should not be read from the source files
  and will remain zero-valued ones. However, this situation does not require
  special processing, since the MPI_File_read_all function automatically stops
  reading the data (without generating any error message)
  when the end of the file is reached.

Input data
Process 0: M = 8 P = 6 Q = 7
File with matrix A: "Adlxmr.tst"
1: -7 3 -7 6 5 -1
7: 2 3 5 -3 -6 -5
13: 6 -4 -1 -7 -4 1
19: -2 2 -4 1 -5 -6
25: -2 6 -7 -6 4 1
31: -5 4 -3 -4 -4 1
37: -6 3 6 2 6 2
43: -7 -5 -7 6 -4 4
File with matrix B: "B2esld.tst"
1: 1 -1 2 5 4 4 4
8: 2 1 -4 7 6 -2 -2
15: 7 7 -7 -4 -2 -4 2
22: 2 5 -1 7 6 6 -4
29: 7 -3 0 -4 -3 3 5
36: 0 -6 -5 -1 0 1 3

Example of right solution
Process 0: NA = 3 P = 6 NB = 2 Q = 7
A0:
-7 3 -7 6 5 -1
2 3 5 -3 -6 -5
6 -4 -1 -7 -4 1
B0:
1 -1 2 5 4 4 4
2 1 -4 7 6 -2 -2
C0:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
Process 1: NA = 3 P = 6 NB = 2 Q = 7
A1:
-2 2 -4 1 -5 -6
-2 6 -7 -6 4 1
-5 4 -3 -4 -4 1
B1:
7 7 -7 -4 -2 -4 2
2 5 -1 7 6 6 -4
C1:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
Process 2: NA = 3 P = 6 NB = 2 Q = 7
A2:
-6 3 6 2 6 2
-7 -5 -7 6 -4 4
0 0 0 0 0 0
B2:
7 -3 0 -4 -3 3 5
0 -6 -5 -1 0 1 3
C2:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0

MPI9Matr8 (1/5)
Acquaintance with the task.
[ENTER - quit, ? - info] PT5Run>

```

Рис. 23: MPI9Matr8
37

Решение заключается в следующем:

```
#include "pt5mpi.hpp"
#include "mpi.h"
#include <cmath>
using namespace std;

int k;           // number of processes
int r;           // rank of the current process

int m, p, q;     // sizes of the given matrices
int na, nb;      // sizes of the matrix bands

int *a_, *b_, *c_; // arrays to store matrices in the master process
int *a, *b, *c;    // arrays to store matrix bands in each process

void Matr1ScatterFile()
{
    char name_a[12], name_b[12];
    int temp[3];
    if (r == 0)
    {
        pt >> temp[0] >> temp[1] >> temp[2];
        pt >> name_a >> name_b;
    }
    MPI_Bcast(temp, 3, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(name_a, 12, MPI_CHAR, 0, MPI_COMM_WORLD);
    MPI_Bcast(name_b, 12, MPI_CHAR, 0, MPI_COMM_WORLD);
    m = temp[0];
    p = temp[1];
    q = temp[2];
    na = ceil((double)m / k);
    nb = ceil((double)p / k);
    MPI_File fa, fb;
    a = new int[na * p];
    b = new int[nb * q];
```

```

    c = new int[na * q];
    for (int i = 0; i < na * p; i++)
        a[i] = 0;
    for (int i = 0; i < nb * q; i++)
        b[i] = 0;
    for (int i = 0; i < na * q; i++)
        c[i] = 0;
    MPI_File_open(MPI_COMM_WORLD, name_a, MPI_MODE_RDONLY, MPI_INFO_NULL, &fa);
    MPI_File_seek(fa, r * na * p * 4, MPI_SEEK_SET);
    MPI_File_read_all(fa, a, na*p, MPI_INT, MPI_STATUSES_IGNORE);
    MPI_File_open(MPI_COMM_WORLD, name_b, MPI_MODE_RDONLY, MPI_INFO_NULL, &fb);
    MPI_File_seek(fb, r * nb * q * 4, MPI_SEEK_SET);
    MPI_File_read_all(fb, b, nb*q, MPI_INT, MPI_STATUSES_IGNORE);
}

void Solve()
{
    Task("MPI9Matr8");
    int rank, size;
    int flag;
    MPI_Initialized(&flag);
    if (flag == 0)
        return;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    k = size;
    r = rank;
    Matr1ScatterFile();
    pt << na << p << nb << q;
    int sz_a = na * p,
        sz_b = nb * q,
        sz_c = na * q;
    for (int i = 0; i < sz_a; ++i)
    {
        pt << a[i];
    }
}

```



```

for (int i = 0; i < sz_b; ++i)
{
    pt << b[i];
}
for (int i = 0; i < sz_c; ++i)
{
    pt << c[i];
}
}
}

```

После решения задачи и запуска терминала ./pt5run MPI9Matr8 cpp будут выведены следующие результаты:

```

-3 -7 1 -4 -7 0 0 -2 -3 -7 5 -2 -2
0 -3 7 5 7 1 -4 2 3 -1 6 0 2
B1:
1 -3 -3 -6 5 -4 -3
4 -5 0 1 4 -7 -2
-1 -2 3 -6 5 2 -7
3 -3 -5 2 4 -3 7
C1:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
Process 2: NA = 3 P = 13 NB = 4 Q = 7
A2:
-7 7 -1 -5 -6 5 6 -2 -6 -6 6 -5 -1
-2 4 -4 1 3 4 -4 7 -7 5 -4 5 4
5 5 4 6 -1 4 -4 -1 -5 -1 -1 -6 -7
B2:
-2 -6 3 1 5 -5 3
-4 2 6 -4 -4 -1 1
-3 4 -4 4 -2 7 -1
3 3 7 6 6 -7 -4
C2:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
Process 3: NA = 3 P = 13 NB = 4 Q = 7
A3:
-3 1 3 -4 6 -1 7 -2 5 2 3 -4 -6
4 0 -5 1 3 -3 -3 -3 3 7 -2 -5 1
0 0 0 0 0 0 0 0 0 0 0 0 0
B3:
-7 6 3 0 -7 -3 5
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
C3:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
MPI9Matr8 (5/5)
The task is solved!
[ENTER - quit, ? - info] PT5Run>

```

Рис. 24: MPI9Matr8

Заключение

В рамках выполнения данной задачи была создана расширенная кроссплатформенная версия электронного задачника Programming Taskbook версии 5. В ходе работы были достигнуты следующие результаты:

- В дополнение к существующему конструктору учебных заданий для языка Free Pascal был разработан новый конструктор для языка C++. Это позволило расширить функциональность задачника и сделать его более универсальным и удобным для пользователей, изучающих различные языки программирования.
- С помощью нового конструктора были перенесены существующие группы заданий, связанных с изучением структур данных стандартной библиотеки шаблонов языка C++ (STL), а также задания, направленные на изучение технологии параллельного программирования MPI. Это обеспечило сохранение всех учебных материалов и позволило их адаптировать под новую версию задачника.
- Все компоненты задачника Programming Taskbook версии 5, ранее реализованные для ОС Windows, были адаптированы для работы в ОС Linux. Это значительно расширило кроссплатформенные возможности задачника, сделав его доступным для пользователей различных операционных систем и обеспечив более широкое применение в образовательных учреждениях и среди индивидуальных пользователей.

По результатам выполненной работы были сделаны доклады на международной научной конференции студентов, аспирантов и молодых учёных «Ломоносов 2024» и на XXXI научной конференции «Современные информационные технологии: тенденции и перспективы развития (СИТО 2024)». [4]

Список литературы

- [1] *Абрамян М. Э.* Об архитектуре универсального электронного задачника по программированию // Информатизация образования и науки. 2015. № 3 (27). С. 134–150.
- [2] *Абрамян М. Э., Лебедев Е. С.* Об одном подходе к реализации кроссплатформенного электронного задачника по программированию / Современные информационные технологии: тенденции и перспективы развития. Материалы XXVII научной конференции. Изд-во ЮФУ, 2020. С. 19–20.
- [3] *Абрамян М. Э., Козак М. В.* Электронный задачник Programming Taskbook: реализация 64-разрядной версии и адаптация к новым средам программирования / Современные информационные технологии: тенденции и перспективы развития. Материалы XXVIII научной конференции. Изд-во ЮФУ, 2021. С. 23–25.
- [4] *Абрамян М. Э., Цюй Жуйчэнь.* Первое сообщение об электронном задачнике Programming Taskbook 5 для Windows и Linux / XXXI научная конференция «Современные информационные технологии: тенденции и перспективы развития» (18–20 апреля 2024 г.). Материалы конференции. Ростов н/Д; Таганрог: Изд-во ЮФУ, 2024. С. 24–27.