# 2 ПРОГРАМНІ ДОКУМЕНТИ

## 2.1 Види програмних документів

ДЕРЖСТАНДАРТ ГОСТ 19.101-77 визначає види програмних документів. Програмними документами є такі документи, що містять відомості необхідні для розробки, супроводу й експлуатації програм [4]. Для можливості посилання на програмні документи у ГОСТах їм привласнені відповідні коди, що для експлуатаційних документів наведені в таблиці 4.

Таблиця 4 – Види експлуатаційних програмних документів

|  |  |  |
| --- | --- | --- |
| Код виду документу | Вид документу | Вміст документу |
| немає  коду | Специфікація | Склад програми та перелік документації на неї |
| 12 | Текст програми | Програмний код з коментарями |
| 13 | Опис програми | Відомості про логічну структуру та функціонування програми |
| 34 | Керівництво оператора | Необхідні відомості для забезпечення роботи оператора при виконанні програми |
| 51 | Програма й методика випробувань | Вимоги, що підлягають перевірці при випробуваннях програми, а також порядок і методи їхнього контролю |
| 81 | Пояснювальна записка | Схеми та опис алгоритмів, опис функціонування програми, обґрунтування прийнятих рішень |

## 2.2 Шифри документів

Для ідентифікації документа ЄСПД ГОСТ 19.103-77 [5] визначає структуру його шифру. На рисунку 1 подано структуру шифру для специфікації, а на рисунку 2 – для інших програмних документів.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | А. В. | | ХХХХХ | | - ХХ | |
| 1. код країни, код організації-розробника | |  | |  | |
| 2. реєстраційний номер програмного виробу | |  | |  | |
| 3. номер редакції (версія програми) | |  | |  | |

Рисунок 1 - Структура шифру специфікації

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ХХ . ХХ . | | ХХХХХ | | -ХХ | | ХХ | ХХ | - ХХ |
| 1. код країни, код організації-розробника | |  | |  | |  | |  |  |
| 2. реєстраційний номер програмного виробу | |  | |  | |  | |  |  |
| 3. номер редакції (версія програми) | |  | |  | |  | |  |  |
| 4. код виду документа | |  | |  | |  | |  |  |
| 5. номер документа даного виду | |  | |  | |  | |  |  |
| 6. номер частини документа | |  | |  | |  | |  |  |

Рисунок 2 - Структура шифру інших програмних

документів (крім специфікації)

Для студентських робіт, присвячених розробці програм та програмних документів, на кафедрі ОТП вирішено, що у загальній частині шифру замість коду країни та організації (перша кодова група) необхідно вказувати назву факультету, та індекс навчальної групи студента, а замість реєстраційного номера програмного виробу (друга кодова група) - п’ятизначний номер залікової книжки. Таким чином, наприклад, шифр документа «Пояснювальна записка», що розроблена студентом з групи КІТ25А із заліковою книжкою з номером 20176 буде такий: КІТ.25А.20176-01 81 01-1

У відповідності до вимог ГОСТ 19.106 – 78 шифр документа проставляється на кожному листі текстової частини документа у верхньому його колонтитулі посередині.

## 2.3 Опис програмних документів

### 2.3.1 Загальні вимоги

Згідно вимог ДЕРЖСТАНДАРТ ГОСТ 19.101-77 [7] програмні документи складаються з:

* вступної частини;
* основної частини;
* додатків.

Вступна частина програмного документу містить:

* аркуш затвердження (див. Додаток А). Це завжди перший лист документу та він не нумерується;
* титульний аркуш (див. Додаток Б), це другий лист документа; його номер - 1, але номер на цьому листі не ставиться;
* анотацію - стисла інформація (до 5 рядків) про зміст документа; пишеться українською, російською та англійською мовами на одному листі. Номер на цьому листі не ставиться;
* зміст (містять всі документи крім специфікації);
* перелік умовних позначень, символів, одиниць, скорочень і термінів (цей лист може бути відсутнім в документі).

***Зверніть увагу*.** Нумеруються всі листи документа починаючи з Титульного листа*. Проставляються номери тільки з другого листа Змісту.* Номер листа записується у верхньому його колонтитулі справа.

На усіх листах (починаючи з анотації) у верхньому колонтитулі посередині проставляється шифр документа.

В розрахунковому завданні необхідно розробити один програмний документ - «Пояснювальна записка».

### 2.3.2 Пояснювальна записка

Код – 81. Оформлення документа Пояснювальна записка регламентуються ЄСПД ГОСТ 19.404–79 .

Документ повинен містити вступ і наступні розділи:

* призначення й область застосування;
* постановку завдання до розробки;
* опис вхідних та вихідних даних;;
* опис складу технічних та програмних засобів;
* список джерел інформації.

У Вступі можна дати, наприклад, таку інформацію:

*Інформаційно-довідкові системи – це такі програмні оболонки, що призначені для керування базами даних і використовуються, наприклад, для ведення статистики, складання каталогів, тощо. Так, інформаційні системи підтримують ведення…..*

Також необхідно навести інформацію про розробляему прикладну галузь.

У розділі ***«Призначення й область застосування»*** вказується призначення програми, дається характеристика області застосування.

Наприклад. *Призначення розробки – оперування даними про ...; дозволяє виконувати пошук за зразком, сортування, додавання даних, їх видалення, тощо. Може використовуватися в …*

У розділі ***«Постановка завдання до розробки»*** поміщають наступне:

* аналітичний огляд інформації з порушеного питання;
* завдання на розробку програми;
* опис застосовуваних математичних методів, обґрунтовується вибір сучасної технології програмування, вказується середовище програмування;
* відповідно до обраного шляху проектування словесно описується обрана структура даних, розроблена структура програми та алгоритм її функціонування. Обґрунтовується коректність алгоритму або окремих його фрагментів;
* аналогічно описуються розроблювані підпрограми.

Аналітичний огляд інформації в даному КП можна почати так:

*В основі функціонування інформаційно-довідкових систем лежить обробка інформації. Режими її обробки можуть бути такими: пакетний, діалоговий, реального часу [2].*

*Діалоговий режим забезпечує безпосередню взаємодію користувача з системою. Ініціатором діалогу може бути як користувач, так і ЕОМ. В останньому випадку на кожному кроці користувачу повідомляється, що треба робити. …В роботі буде реалізовано….*

*Дані, що обробляються, в оперативній пам’яті можуть зберігатися у вигляді масиву або лінійного (одно- чи двонаправленого) списку[1] . …*

*До переваг масиву можна віднести …, до недоліків - ….*

*До переваг списку можна віднести …, до недоліків - ….*

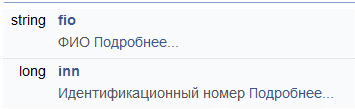
*Виходячи з …в розроблюваній програмі для представлення даних буде реалізовано ….*

В завданні на розробку, виходячи з аналітичного огляду, конкретизується перелік завдань, щодо розробки.

При виборі мови і середовища програмування можна зазначити таке:

*Об’єктно-орієнтоване програмування має такі переваги: …. Мова програмування … забезпечує …. Тому для розробки програми обрано мову програмування …, а середовище програмування - ….*

При розробці структур даних слід виходити з заданого об’єкту та його властивостей. Тут можна запропонувати, наприклад, таке:



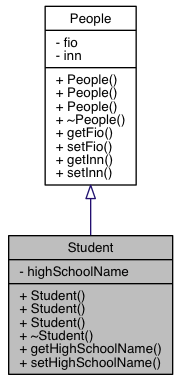
*Рисунок 1* – *Поля класу People*

*Абстрактний клас People – базовий. Його наслідують клас Student, додаткові поля якого наведено на рис. 2…*



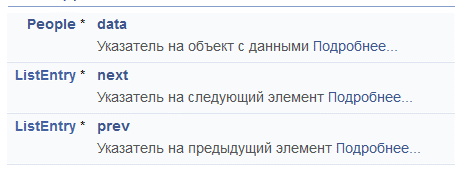
*Рисунок 2* – *Додаткаві поля класу Student – нащадка класу People*

*На рис. 3 показано внутрішню структуру та відносини розроблених класів у вигляді UML діаграми.*



*Рисунок 3* – *Схема ієрархії розроблених класів*

*Дані про всіх студентів будуть заноситися до списку. Для цього треба ввести структуру ListEntry з такими полями, як у рис. 4.*



*Рисунок 4* – *Поля* структури *ListEntry*

*Для представлення списку розроблено клас List з закритими атрибутами та відкритими методами, що наведені на рис. 5.*

*Для роботи з списком, розроблено клас ListProcessor, методи якого представлені на рис. 6.*

При опису структури програми можна вказати, наприклад, таке:

*Програма складається з таких модулів:*

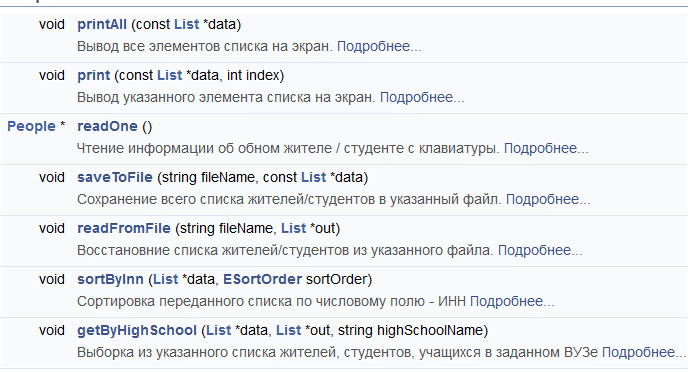
* *інтерфейсного, який містить функцію main і функції реалізації діалогу;*
* *модуля, що містить опис класу People;*
* *модуля, що містить опис класу Student;*

*. . .*

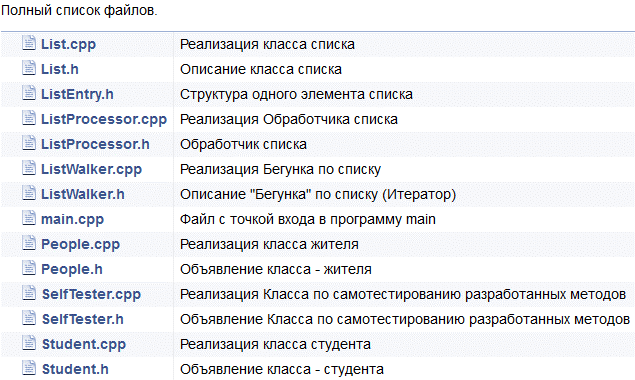
*На рис. 7 подано структуру проекту розробленого програмного продукту.*



*Рисунок 5* – *Поля та методи класу List*



*Рисунок 6* –М*етоди класу ListProcessor*



*Рисунок 7 – Структура проекту*

При опису функцій слід висвітлити таке:

* призначення;
* посилання на схему алгоритму функції (якщо є);
* порядок роботи.

При опису головної функції *main* можна зазначити, наприклад, таке:

*Функція main координує роботу всіх функцій програми.*

*Схема алгоритму подана на рис.x.*

*Порядок роботи: В циклі видається меню, вводиться команда оператора (блок 1) , яка розпізнається в блоці 2 і залежно від значення команди керування передається відповідній функції. Програма припиняє роботу після введення команди «8» .*

В розділі ***«Опис вхідних та вихідних даних»*** подається обґрунтований вибір методу організації вхідних і вихідних даних, перераховується необхідний набір і джерело вхідних даних, спосіб їх одержання розроблювальною програмою.

Наприклад*, Вхідні дані можуть вводитися оператором з клавіатури в такій послідовності …, або з типізованого файлу.*

Для вихідних даних аргументується вибір носія для їх зберігання та спосіб відображення для користувача.

*Вихідні дані зберігаються в типізованому файлі. Типізований файл обрано тому, що у порівнянні з текстовим файлом він має такі переваги …. До недоліків слід віднести таке ….*

В розділі ***«Список джерел інформації»*** наводять перелік науково-технічних публікацій, нормативно-технічних документів і інших науково-технічних матеріалів. Докладна інформація про надання переліку джерел подана в п. 6.2.9 даних вказівок.

*Увага.* В разі необхідності ГОСТ дозволяє об’єднувати окремі розділи документу і додавати інші. В Додатку Г подано приклад змісту Пояснювальної записки.

Приклад аркуша затвердження пояснювальної записки подано у додатку А, титульного аркуша – додатку Б, змісту – додаткуВ.

### 2.3.3 Оформлення схем програмної документації

Алгоритм – це будь-яка обчислювальна процедура, що описує несуперечливий хід виконання завдання від входу до отримання результату. Алгоритм відображає послідовність обчислювальних кроків, які дозволяють перетворити вхідні дані у вихідні.

При представленні алгоритму у схематичному вигляді використовуються спеціальні графічні символи. У стандарті ЄСПД ГОСТ 19.701-90 визначені графічні символи, що призначені для використання в документації по обробці даних, і правила їх використання [13]. Якщо фрагменти схем описуються в текстовому документі, наприклад, у пояснювальній записці, в описі програми й т.п. з метою підкреслити важливі або оригінальні логічні особливості розроблювального виробу, то ці фрагменти вставляють у текст у вигляді рисунків з підрисуночним підписом. Якщо схема повністю описує всю розробку, то вона представляється на кресленнях у рамках з основним написом за формою 2. У схемах лініями відображають напрямок передачі даних при рішенні поставленого завдання, символами процесу відзначаються етапи обробки даних.

# 3 ОСНОВНІ ВИМОГИ ДО ОФОРМЛЕННЯ ТЕКСТОВИХ ДОКУМЕНТІВ

## 3.1 Загальні вказівки

Згідно вимог стандарту СТВУЗ–ХПИ-3.01–2008 текстові документи оформляються на аркушах формату А4 (210\*297мм) [3]. При необхідності, (зображення схем, таблиць, рисунків) можна використовувати аркуші формату А3 (297\*420мм), складені до формату А4. Документи виконуються за допомогою комп’ютерної техніки на одній стороні листа білого паперу, на українській мові.

Розмір полів: верхнє – 20мм, ліве – 25мм, нижнє – 15мм, праве – 10мм. Друк виконується шрифтом Times New Roman текстового редактора Word розміру 14 пунктів і полуторним міжрядковим інтервалом.

Рисунки (у тому числі схеми алгоритмів) виконуються машинним способом, олівцем, або чорною пастою.

## Правила оформлення текстових документів

### 3.2.1 Нумерація сторінок

Нумерація сторінок документа наскрізна (включаючи додатки). Номера сторінок проставляються арабськими цифрами в правому верхньому куті без крапки. Починають проставлять номера сторінок з другого листа Змісту.

### 3.2.2 Побудова тексту

Написання текстових документів потрібно виконувати за вимогою ДСТУ 3008 –95 та стандарту СТВУЗ-ХПИ-3.01-2008 [1, 3].

Текстовий документ може містити суцільний текст з абзацами і текст, розбитий на графи (специфікації, таблиці, відомості і т.п.). Не допускаються будь-які прикраси тексту зміною шрифту, підкресленням слів, фраз, чи назв елементів рубрикації, застосуванням різнобарвних букв і т.д.

Структурні елементи: «ЗМІСТ», «ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ», «ВСТУП», «ВИСНОВОК», «СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ІНФОРМАЦІЇ» не нумерують, а їхні назви служать заголовками структурних елементів.

Кожен розділ тексту повинний починатися з нової сторінки і мати порядкову нумерацію в межах усього документу арабськими цифрами без крапки, наприклад: 1, 2, 3 і т.д.

Нумерація підрозділу складається з номера розділу і порядкового номера підрозділу, розділених крапкою. Після номера підрозділу крапку не ставлять, наприклад: 1.1, 1.2 і т.д.

Якщо в тексті є пункти, то в межах кожного підрозділу вони повинні мати порядкову нумерацію і їхній номер складається з номера розділу, номера підрозділу, пункту, розділених крапкою. Після номера пункту крапку не ставлять, наприклад: 1.1.1, 1.1.2 і т. д.

Підпункти тексту нумеруються окремо в межах кожного пункту. Їх номера складаються з номера розділу, порядкових номерів підрозділу, пункту і підпункту, розділених крапкою, наприклад: 2.1.1.1, 2.1.1.2, 2.1.1.3 і т.д. Після номера підпункту крапку не ставлять.

Кожний розділ, пункт і підпункт повинні мати короткий заголовок, що відповідає змісту. Заголовки розділів розташовують посередині рядка і записують прописними буквами, а підрозділів, пунктів і підпунктів починають з абзацного відступу і записують малими (крім першої прописної) жирними літерами. Заголовки не підкреслюють і крапку наприкінці не ставлять. Перенос слів у заголовках не допускається. Якщо заголовок складається з двох чи більш частин, їх розділяють крапкою.

Відстань між заголовком і наступним текстом повинна бути такою, як в тексті, а та між заголовком і попереднім – 12 пунктів. Не допускається розміщати найменування розділу, підрозділу, пункту і підпункту в нижній частині сторінки, якщо після нього розташований тільки один рядок тексту (заборона «висячих рядків»).

### 3.2.3 Виклад тексту

Виклад тексту повинен бути коротким і чітким. Текст викладають у відповідності зі стандартом і технічними умовами, прийнятими в науково-технічній літературі, тобто виклад тексту ведеться від третьої особи в дійсному нахиленні, вживаючи дієслова невизначеної форми. Наприклад, замість «приймаю» або «приймаємо» треба писати «приймаються», «дані розрахунку приводяться...», «у розділі розглядається...».

Скорочення слів у тексті і підписах під ілюстраціями, як правило, не допускається. ГОСТ 2.316-81 визначив виключення, що дозволяють скорочувати окремі слова і словосполучення тексту. Основні виключення:

* і т.п., і т.д., і ін., ін. – тільки наприкінці фрази;
* див., табл., рис., стор., вип., журн., л. (лист), п. (пункт), розд. (розділ), ст. (стаття) – при посиланнях і виносках у суцільному тексті.

Якщо в документі прийнята особлива система скорочення слів або найменування, то в документі приводять перелік прийнятих скорочень.

Оборот «від…до» у тексті варто уникати, його можна замінити через межу 5-10 або крапки 40...60. Абстрактні числа до 9 пишуться словами, понад 9 – цифрами (три криві; 10 розділів і т.д.).

Числа з розмірністю пишуться тільки цифрами, наприклад, 3 км, 5 грн. і т.п. Дробові значення проставляються тільки цифрами – «1/4 ч».

Числові значення в тексті пишуться тільки від 0.1 до 1000. Для більших значень вводять кратні одиниці (мега – 106; гіга – 109; тера – 1012; пета – 1015; еска – 1018), а для меньших – дольові (санти – 10-2; мілі – 10-3; мікро – 10-6; нано– 10-9; піко– 10-12;фемто- 10-15; атто-10-18).

Наприклад, 110 тера байт, 0.15 А (0.15 ампер).

### 3.2.4 Оформлення ілюстрацій

Ілюстрації можуть бути розташовані по тексту документа відразу після першого посилання на них або на наступній сторінці. Крім того, вони можуть бути подані в додатках. Усі наведені в документі ілюстрації позначають словом «Рисунок» і нумерують арабськими цифрами в межах усього тексту (при їх невеликій кількості) або кожного розділу (при їх великій кількості). У випадку, коли ілюстрації нумерують у межах розділу, номер ілюстрації складається з номера розділу і порядкового номера ілюстрації, розділених крапкою. Номер ілюстрації в тексті вказують так: Рисунок 1 або Рисунок 2.1. (перша ілюстрація другого розділу).

При посиланні в тексті на ілюстрацію треба вказати її повний номер, наприклад, рис.1, рис.1.5. Повторні посилання на раніше згадані ілюстрації дають зі скороченням слова «дивися», наприклад, (див. рис.6.2). У випадку наявності в документі однієї ілюстрації вона в обов’язковому порядку підлягає нумерації з обліком викладених вище положень.

Ілюстрації повинні мати тематичну назву, що починається з прописної букви, без крапки наприкінці і розміщується під ілюстрацією. При необхідності ілюстрації доповнюють пояснювальними даними.

Наприклад: Рисунок 1.15 – Схема алгоритму програми

### 3.2.5 Побудова таблиць

У табличній формі подають інформацію, що не піддається відтворенню іншими способами (у вигляді схем, графіків та ін.).

Таблицю розташовують після першого згадування про неї або на наступній сторінці. У документі таблиці повинні відповідати вимогам ДСТУ 3008-95. При наявності в тексті декількох таблиць їх нумерують арабськими цифрами. Напис типу «Таблиця 2» або «Таблиця 1.2» (друга таблиця першого розділу) поміщають ліворуч над таблицею без лапок і крапки.

На всі таблиці повинні бути посилання в тексті, наприклад, (табл. 3). При повторному посиланні – (див. табл. 3). Якщо в тексті одна таблиця, то їй привласнюють номер відповідно до викладених вище вимогам.

Таблиця в обов’язковому порядку повинна мати тематичний заголовок, який необхідно виконувати малими літерами (крім першої прописної) і поміщати над таблицею разом зі словом Таблиця. Номер таблиці та її назва відокремлюються символом «дефіс».

Наприклад: Таблиця 2 – Призначення змінних

### 3.2.6 Написання формул

Формули набираються в редакторі формул Microsoft Equation 3.0 з такими настроюваннями: основний текст 14 пунктів, великий індекс -20 пунктів, дрібний індекс 7 пунктів, великий символ - 20 пунктів. Латинські символи - похилі, грецькі й кирилиця - прямі символи.

Формули, на які є посилання в тексті, повинні нумеруватися в межах розділу арабськими цифрами. Номер формули повинен складатися з номера розділу й порядкового номера формули, розділених крапкою. Номер формули варто брати в дужки й поміщати на правому полі на рівні нижнього рядка формули, до якої вона відноситься.

При посиланні в тексті на формулу необхідно вказувати її повний номер у дужках, наприклад: “У формулі (1.2)”.

### 3.2.7 Оформлення приміток

У примітках до тексту і таблиць указують тільки довідкову інформацію, що її пояснює. Примітки розташовують безпосередньо після тексту, таблиці, ілюстрації, до яких вони мають відношення. На кресленнях примітки не дають.

Слово «Примітка» пишуть із прописної букви з абзацного відступу, не підкреслюють, після даного слова ставлять крапку і з прописної букви в тім же рядку дають текст примітки.

Якщо приміток декілька, то після слова «Примітки» ставлять двокрапку. Примітки нумерують арабськими цифрами з крапкою після номеру. Кожну примітку записують з нового рядка з абзацу. Текст примітки починається з прописної букви. Наприклад,

Примітки:

1 ...

2 ...

### 3.2.8 Посилання на нормативну літературу

Посилання на нормативну літературу (стандарти, правила, технічні умови, інструкції) і інші джерела (книги, статті, патенти), у тексті документа необхідно наводити у випадку, якщо в них містяться відомості, що підтверджують обґрунтованість прийнятих рішень, методик розрахунків, вибір формул, коефіцієнтів, нормативних величин. При посиланні потрібно вказувати в квадратних дужках порядковий номер приведеного наприкінці документа списку літератури, наприклад [1], [3,5,7], [5,107-129 с.].

### 3.2.9 Оформлення списка джерел інформації

Перелік літературних джерел повинний бути поданий наприкінці документа, починаючи з нової сторінки, під заголовком «СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ». Він включає джерела, якими користався студент у процесі виконання роботи і на які є посилання в тексті.

Літературні джерела в списку подаються в послідовності їх згадування у тексті документа. Порядкові номери описів у переліку є посиланнями в тексті (номерні посилання). Відомості про літературне джерело приводять на тій мові, на якій воно опубліковане.

Бібліографічне посилання містить наступні елементи:

а) для книг: прізвище і ініціали автора (або авторів); назва книги; місце видання – однією великою буквою «М.», «К», «Л.» відповідно для міст Москви, Києва і Ленінграда (стара назва) і цілком назва інших міст без слова «місто»; видавництво; рік видання і загальна кількість сторінок джерела, наприклад:

1. Шилдт, Герберт. Полный справочник по С++, 4-е издание.: Пер. с англ.: – М.: Изд. дом “Вильямс” , 2004.- 800 с.

б) для журнальних статей: прізвище і ініціали автора (авторів), повна назва статті, назва журналу, рік видання, випуск номер журналу, кількість сторінок статті, наприклад,

2. Бабій А.С. Програмна система // Вестник НТУ «ХПИ». Сборник научных трудов. Тематический выпуск «Информатика и моделирование”. – Харьков: НТУ «ХПИ». - 2007.- № 19.-с. 12 - 16.

г) для методичних вказівок: назва вказівки, прізвища і ініціали авторів, місце видання, видавництво, рік видання, кількість сторінок, приклад:

3. Методические указания к выполнению лабораторных работ по курсу «Архитектура ЭВМ». Составители: Поворознюк А.И., Зыков И.С. – Харьков: ХПИ, 2000 – 79 с.

### 3.2.10 Оформлення додатків

Додатки оформляють як продовження документу на наступних його аркушах. У додатках, що поміщають наприкінці пояснювальної записки, можуть бути включені матеріали допоміжного характеру, такі як: додаткові ілюстрації та таблиці, проміжні математичні докази, формули, розрахунки, тексти комп’ютерних програм, структури і змісти вхідних і вихідних даних (файлів, копій екранів з результатами роботи програми в текстовому і графічному вигляді), вид інтерфейсу користувача, інструкції, методики тестування.

Кожен додаток починається з нової сторінки і має заголовок, що записують малими літерами з першої прописної, симетрично щодо тексту. Посередині рядка над заголовком малими літерами, крім першої прописної, вказують слово «Додаток\_\_\_» і прописну букву, що позначає додаток.

Додатки варто позначати послідовно прописними буквами українського алфавіту, за винятком Є, З, І, Ї, Й, О, Ч, Ь.

Приклад:

ДОДАТОК А

Структура вхідних та вихідних даних

Текст кожного додатка може бути розбитий на розділи, підрозділи, пункти і підпункти з відповідною нумерацією згідно зазначених вище вимог. Перед кожним номером проставляють позначення додатка (букву) і крапку, наприклад, А.2 – другий розділ додатка А; В.3.1 – підрозділ 3.1 додатка В; Д.4.1.2 – пункт 4.1.2 додатка Д и т.д.

Один додаток позначається як додаток А и при наявності в ньому тільки однієї таблиці, ілюстрації, формули чи рівняння в обов’язковому порядку їх нумерують, наприклад, Рис. А.1, Таблиця А.1, формула (А.1).

Нумерація сторінок записки і додатків повинна бути наскрізна, причому в змісті пояснювальної записки необхідно перелічити всі додатки.

При посиланнях у тексті додатка на ілюстрації, таблиці, формули чи рівняння рекомендується писати:«на рис. А.2»;«у таблиці Б.3» або «у табл.Б.3»; «по формулі (В.1)»; «у рівнянні (Д.5)».

# СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. ДСТУ 3008–95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. – Чинний від 01.01.96 – Київ: Держстандарт України, 1995. – 37 с.
2. ГОСТ 7.1–2003, IDT : ДСТУ ГОСТ 7.1-2006. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання. – Чинний з 2007–07–01. – К. : Держспоживстандарт України, 2007.
3. СТВУЗ-ХПИ-3.01-2010. Текстові документи у сфері навчального процесу. Загальні вимоги до виконання. – Введ. 01.07.10 г.
4. ГОСТ 19.101–77. Единая система программной документации. Виды программ и программных документов. – Введ. с 01.01.80.
5. ГОСТ 19.103–77. ЕПКД. Обозначение программ и программных документов. – Введ. с **01.01.80.**
6. ГОСТ 2.106–96. **Единая система конструкторской документации. Текстовые документы.** – **В**вед. с 13.11.96.
7. ГОСТ 19.105–78. ЕСПД. Общие требования к программным документам. – Введ. 01.01.1980
8. ГОСТ 19.202–78. ЕСПД. Спецификация. Требования к содержанию и оформлению. – Введ. 01.01.80.
9. ГОСТ 19.401–78. ЕСПД. Текст программы. Требования к содержанию и оформлению. – Введ. с 01.01.80.
10. ГОСТ 19.402–78. ЕСПД. Описание программы. Требования к содержанию и оформлению. – Введ. c 01.01.80.
11. ГОСТ 19.505–79. ЕСПД. Руководство оператора. Требования к содержанию и оформлению. – Введ. с 01.01.80.
12. ГОСТ 19.301–79. ЕСПД. Программа и методика испытаний. Требования к содержанию и оформлению. – Введ. с 01.01.81.
13. ГОСТ 19.701–90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. – Введ. с 01.01.92.

ДОДАТОК А

Приклад аркуша затвердження пояснювальної записки

МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ «ХПІ»

Кафедра «Обчислювальна техніка та програмування»

Розрахункове завдання з програмування

Тема: «РОЗРОБКА ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ»

Пояснювальна записка

КІТ.15А.20176-01 81 01-1 -АЗ

Розробники

###### Виконав:

студент групи КІТ-15А

\_\_\_\_\_\_\_\_\_**\_**\_**\_\_\_\_**/ Іванов О.А./

Перевірив:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/доц. Далека В.Д./

Харків 2015

# ДОДАТОК Б

# Приклад титульного аркуша пояснювальної записки

ЗАТВЕРДЖЕНО

КІТ.15А.20176-01 80 01-1 -АЗ

Розрахункове завдання з програмування

Тема: «РОЗРОБКА ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ»

Пояснювальна записка

КІТ.15А.20176-01 81 01-1

Аркушів \_32\_

Харків 2015

# 

# ДОДАТОК В

# Приклад змісту пояснювальної записки

ЗМІСТ

|  |  |
| --- | --- |
| Вступ…............................................................................................................  1 Огляд літературних джерел……….……..…………………………..…...  1.1 Поняття «Інформаційна система» .........................................................  1.2 Постановка завдання до розробки ........................................................  2 Розробка інформаційно-довідкової системи…………….......................  2.1 Розробка структур даних .......................................................................  2.2 Розробка структури програми................................................................  2.3 Розробка алгоритмів програми..............................................................  2.3.1 Розробка алгоритму головної функції................................................  2.3.2 Розробка методів класу *People* ...........................................................  2.3.3 Розробка методів класу *Student* …….................................................  2.3.4 Розробка методів класу *List* ………...................................................  2.3.5 Розробка методів класу *ListProcessor* ...............................................  Висновки….....................................................................................................  Список джерел інформації……………………………................................  Додаток А. Схеми алгоритмів розроблених методів.................................  Додаток Б. Текст програми...........................................................................  Додаток В. Результати роботи програми..................................................... | 3  4  4  6  7  7  7  8  9  9  10  11  12  13  14  15  20  30 |

# 

# ДОДАТОК Г

# Приклад додатка «Текст програми»

1 Файл List.h. Опис класу List

/\*\*

**\* @file List.h**

\* Описание класса списка.

\*

\* @author davs

\* @version 0.0.1

\* @date 26-nov-2015

\*/

#ifndef COURSEEXAMPLE\_LIST\_H

#define COURSEEXAMPLE\_LIST\_H

#include "ListEntry.h"

#include "ListWalker.h"

/\*\*

\* Двухсвязанный список. Сожержит головной и хвостовой элементы, между которыми находятся фактичекие данные.

\* Данный класс имеет только конструктор по-умолчанию, ибо использовать другие конструторы здесь не имеет смысл

\*/

class List {

private:

ListEntry \*head; ///< указатель на первый элемент

ListEntry \*tail; ///< указатель на последний элемент

public:

/\*\*

\* Конструктор по-умолчанию. Инициализирует голову и хвост соответствующими элементами типа ListEntry.

\* При этом, в связи с тем, что мы имеем дело с двухсвязанным списком, следующий элемент от головы является

\* хвост, а предыдущий элемент хвоста - голова

\*/

List();

/\*\*

\* Деструктор. Освобождает память, выделенную конструтором для головы и хвоста.

\* Если в список были добавлены элементы, то они также должны быть очищены (в первую очередь - до очистки

\* граничных элементов)

\*/

~List();

/\*\*

\* Добавление объекта в заданную позицию. Объектом может быть как родительский класс (Житель),

\* так и дочерний (Студент)

\* Если позиции не существует - добавляет элемент в конец списка (перед хвостовым элементом)

\* @param data объект для добавления

\* @param position позиция для добавления

\*/

void addEntry(const People\* data, int position);

/\*\*

\* Удаление объекта по заданной позиции.

\* Если позиции не существует - удаляет последний элемент из списка;

\* Если список пустой - то ничего не делаем.

\* @param position позиция для удаления

\*/

void removeEntry(int position);

/\*\*

\* Получение Итератора для хождения по списку.

\* @return итератор, ссылающийся по-умолчанию, на головной элемент списка

\* @see ListWalker

\*/

ListWalker getNewWalker();

/\*\*

\* Очистка списка. После удаления остаются только головной и хвостовой элементы

\*/

void clear();

/\*

\* Получение объекта по заданной позиции. Если элемента по заданной позиции не существует - вызывает ошибку;

\* @param index позиция элемента для получения

\* @return ссылка на элемент из списка с указанным индексом

\*/

People& operator[] (size\_t index);

};

#endif //COURSEEXAMPLE\_LIST\_H

2 Файл List.cpp. Реалізація класу List

/\*\*

**\* @file List.cpp**

\* Реализация класса списка.

\*

\* @author davs

\* @version 0.0.1

\* @date 26-nov-2015

\*/

#include "List.h"

List::List() {

head = new ListEntry();

tail = new ListEntry();

}

List::~List() {

// [реализация упущена] удаление промежуточных элементов

delete head;

head = NULL;

delete tail;

tail = NULL;

}

void List::addEntry(const People \*data, int position) {

// [реализация упущена] добавление элемента в указанную позицию

}

void List::clear() {

// [реализация упущена] очистка списка

}

void List::removeEntry(int position) {

// [реализация упущена]

}

ListWalker List::getNewWalker() {

return ListWalker(head);

}

People &List::operator[](size\_t index) {

// [реализация упущена]

}

3 Файл ListEntry.h. Структура одного елемента списку

/\*\*

**\* @file ListEntry.h**

\* Структура одного элемента списка

\* @author davs

\* @version 0.0.1

\* @date 26-nov-2015

\*/

#ifndef COURSEEXAMPLE\_LISTENTRY\_H

#define COURSEEXAMPLE\_LISTENTRY\_H

#include "People.h"

/\*\*

\* Элемент двухсвязного списка. Характеризуется ссылками на предыдущий, следующий элементы и на класс с данными.

\* Если элемент находится в начале списка ("голова"), то указатель на предыдущий элемент равен NULL.

\* Если элемент находится в конце списка ("хвост"), то указатель на следующий элемент равен NULL.

\*/

struct ListEntry {

People \* data; ///< Указатель на объект с данными

ListEntry \* next; ///< Указатель на следующий элемент

ListEntry \* prev; ///< Указатель на предыдущий элемент

};

#endif //COURSEEXAMPLE\_LISTENTRY\_H

1. Файл ListProcessor.h. Обробник списка

/\*\*

**\* @file ListProcessor.h**

\* Обработчик списка

\* @author davs

\* @version 0.0.1

\* @date 27-nov-2015

\*/

#ifndef COURSEEXAMPLE\_LISTPROCESSOR\_H

#define COURSEEXAMPLE\_LISTPROCESSOR\_H

#include "List.h"

/\*\*

\* Порядок сортироки

\*/

enum class ESortOrder {

ASC, ///< по-возрастанию

DESC ///< по-убыванию

};

/\*\*

\* Обработчик связного списка.

\* Содержит методы по работе с списком согласно заниданию к РГЗ

\*/

class ListProcessor {

public:

/\*\*

\* Вывод все элементов списка на экран.

\* @param data указатель на список, элементы которого должны быть выведены на экран.

\*/

void printAll(const List \*data);

/\*\*

\* Вывод указанного элемента списка на экран. Если элемент с указанной позицией не существует,

\* на экрна ничего не будет выведено.

\*

\* @param data указатель на список, элемент которого должен быть выведен на экран.

\* @param index позиция выводимого элемента в списке

\*/

void print(const List\* data, int index);

/\*\*

\* Чтение информации об обном жителе / студенте с клавиатуры. С клавиатуры задается тип создаваемого объекта,

\* на основе которго вводятся либо поля жителя, либо поля студента.

\*

\* @return сформированный объект из данных с клавиатуры.

\*/

People\* readOne();

/\*\*

\* Сохранение всего списка жителей/студентов в указанный файл. При этом, информация о головном и хвостовом

\* элементах списка в файл не попадает.

\*

\* @param fileName имя файла (или путь + имя файла), в который будет сохранена информация.

\* @param data список, который должен быть сохранен

\*/

void saveToFile(string fileName, const List\* data);

/\*\*

\* Восстановние списка жителей/студентов из указанного файла. При этом вся текущяя информация о жителях стирается

\*

\* @param fileName имя файла (или путь + имя файла), из которого будет прочитана / восстановлена информация.

\* @param out список, в который должна быть восстановлена информация.

\*/

void readFromFile(string fileName, List\* out);

/\*\*

\* Сортировка переданного списка по числовому полю - ИНН

\* @param data Список, поля которых следует отсортировать

\* @param sortOrder порядок сортировки.

\*/

void sortByInn(List\* data, ESortOrder sortOrder);

/\*\*

\* Выборка из указанного списка жителей, студентов, учащихся в заданном ВУЗе

\* @param data Начальный список, содержащий обычных жителей и студентов, обучающихся в разных ВУЗах

\* @param highSchoolName Имя ВУЗа, список студентов которых мы хотим выбрать

\* @param out Результирующий список, в который занесутся данные. Все существующие данные этого списка будут удалены!!

\*/

void getByHighSchool(List\* data, List\* out, string highSchoolName);

};

#endif //COURSEEXAMPLE\_LISTPROCESSOR\_H

1. Файл ListProcessor.cpp. Реалізація обробника списка
2. /\*\*

**\* @file ListProcessor.cpp**

\* Реализация Обработчика списка

\* @author davs

\* @version 0.0.1

\* @date 27-nov-2015

\*/

#include "ListProcessor.h"

void ListProcessor::printAll(const List \*data) {

// [реализация упущена]

}

void ListProcessor::print(const List \*data, int index) {

// [реализация упущена]

}

People \*ListProcessor::readOne() {

// [реализация упущена]

return nullptr;

}

void ListProcessor::saveToFile(string fileName, const List \*data) {

// [реализация упущена]

}

void ListProcessor::readFromFile(string fileName, List \*out) {

// [реализация упущена]

}

void ListProcessor::sortByInn(List \*data, ESortOrder sortOrder) {

// [реализация упущена]

}

void ListProcessor::getByHighSchool(List \*data, List \*out, string highSchoolName) {

// [реализация упущена]

}

1. Файл ListWalker.h. Опис ітератора

/\*\*

**\* @file ListWalker.h**

\* Описание "Бегунка" по списку (Итератор).

\* @author davs

\* @version 0.0.1

\* @date 03-dec-2015

\*/

#ifndef COURSEEXAMPLE\_LISTWALKER\_H

#define COURSEEXAMPLE\_LISTWALKER\_H

#include "People.h"

#include "ListEntry.h"

/\*\*

\* Итератор для списка. <i>(В рамках РГЗ имплементация не обязательна, но крайне желательна).</i>

\* Позволяет "ходить" по списку в любом направлении, изменять содержимое списка / получать значения в процессе "хождения".

\* Используется для повышения производительности при прохождении по списку, т.к. позволяет использовать "умный" подход при

\* доступе к произвольному элементу, а не проходить к нему с начала списка.

\*/

class ListWalker {

ListEntry \*current; ///< указатель на текущий элемент из списка

public:

/\*\*

\* Инициализация итератора.

\* @param newCurrent указатель на <b>головной элемент списка</b>, с которым прийдется работать.

\*/

ListWalker(ListEntry \* newCurrent);

/\*\*

\* Удаление текущего (current) элемента из списка. Если мы находимся на голове - то ничего не делаем.

\*/

void deleteCurrentEntity();

/\*\*

\* Получение объекта, который относится к текущему элементу списка. Если находимся на голове списка - ничего не делаем.

\* @return текущий объект списка

\*/

People\* getCurrentEntry();

/\*\*

\* Обновление текущего элемента. Если находимся на голове списка - ничего не делаем.

\*/

void updateCurrentEntry(People\* newData);

/\*\*

\* Вставка элемента после текущего

\* @param newData новый элемент для добавления

\*/

void insertAfterCurrent(People \*newData);

/\*\*

\* Перемещает указатель текущего элемента списка вперед на один элемент.

\* Если мы находимся в конце списка (сразу перед хвостом) - то ничего не выполняем.

\*/

void moveCurrentElementForward();

/\*\*

\* Перемещает указатель текущего элемента списка назад на один элемент.

\* Если мы находимся в начале списка (голова или первый после головы) - то ничего не выполняем.

\*/

void moveCurrentElementBackward();

/\*\*

\* Получение индекса текущего элемента. Если находимся на голове - возвращает -1, иначе - индекс относительно головы,

\* начиная с 0.

\* @return индекс текущего элемента

\*/

size\_t getCurrentIndex();

};

#endif //COURSEEXAMPLE\_LISTWALKER\_H

7 Файл ListWalker.cpp. Реалізація ітератора

/\*\*

**\* @file ListWalker.cpp**

\* Реализация Бегунка по списку

\* @author davs

\* @version 0.0.1

\* @date 03-dec-2015

\*/

#include "ListWalker.h"

ListWalker::ListWalker(ListEntry \* newCurrent) {

current = newCurrent;

}

void ListWalker::deleteCurrentEntity() {

// [реализация упущена] удаление элемента

}

People \*ListWalker::getCurrentEntry() {

// [реализация упущена] получение элемента по индексу

return nullptr;

}

void ListWalker::moveCurrentElementForward() {

// [реализация упущена]

}

void ListWalker::moveCurrentElementBackward() {

// [реализация упущена]

}

void ListWalker::updateCurrentEntry(People \*newData) {

// [реализация упущена]

}

void ListWalker::insertAfterCurrent(People \*newData) {

// [реализация упущена]

}

size\_t ListWalker::getCurrentIndex() {

// [реализация упущена]

return -1;

}

8 Файл People.h. Об’ява класу People

/\*\*

\* @file people.h

\* Объявление класса - жителя

\* @author davs

\* @version 0.0.1

\* @date 26-nov-2015

\*/

#ifndef COURSEEXAMPLE\_PEOPLE\_H

#define COURSEEXAMPLE\_PEOPLE\_H

#include <string>

using namespace std;

/\*\*

\* Житель страны, характеризующийся ФИО и Идентификационным номером.

\*/

class People {

private:

string fio; ///< ФИО

long inn; ///< Идентификационный номер

public:

/\*\*

\* Конструктор по-улолчанию. Инициализация полей пустыми начальными значениями.

\*/

People();

/\*\*

\* Конструктор с параметрами. Инициализация характристик жителя переданными параметрами

\* @param newFio ФИО жителя

\* @param newInn идентификационный номер (ИНН) жителя

\*/

People(string newFio, long newInn);

/\*\*

\* Конструктор копирования. Инициализируем жителя теми же характеристиками, что и переданный.

\* @param newPeople человек, по образу и подобию которого будем создавать текущего жителя

\*/

People(const People& newPeople);

/\*\*

\* Деструктор. Очиска выделенной в конструкторе памяти.

\* В связи с тем, что в конструкторе ничего не выделялось, деструктор - пустой

\*/

virtual ~People();

/\*\*

\* Получение ФИО жтеля

\* @return текущее ФИО жителя

\*/

const string getFio() const;

/\*\*

\* Установка новой фамилии жителю

\* @param fio Новая фамилия

\*/

void setFio(const string &fio);

/\*\*

\* Получение ИНН жтеля

\* @return текущий ИНН жителя

\*/

long getInn() const;

/\*\*

\* Установка нового идентификационного номера жителю

\* @param inn Новый ИНН

\*/

void setInn(long inn);

/\*\*

\* Сравнение двух людей по ИНН

\* @param p1 человек, которого сравниваем (левый операнд)

\* @param p2 человек, с которым сравниваем (правый операнд)

\* @return <code>true</code>, если ИНН левого операнда меньше ИНН второго операнда, иначе - <code>false</code>

\*/

friend bool operator < (const People& p1, const People& p2);

/\*\*

\* Сравнение двух людей по ИНН

\* @param p1 человек, которого сравниваем (левый операнд)

\* @param p2 человек, с которым сравниваем (правый операнд)

\* @return <code>true<code>, если ИНН левого операнда больше ИНН второго операнда, иначе - <code>false</code>

\*/

friend bool operator > (const People& p1, const People& p2);

/\*\*

\* Вывод информации текущего человека в поток вывода. Функция рассчитана на вывод на экран, по этому имеет

\* следующий формат: '[ИНН (длиной 15 символов)]: [FIO]'

\* @param out поток вывода. Так как функция рассчитана на вывод на консоль, то в качестве данного параметра

\* передается переменная <code>cout</code>

\* @param people человек, информация о котором передаем в поток вывода

\* @return поток вывода для дальнейшего вывода данных

\*/

friend ostream& operator << (ostream& out, const People& people);

/\*\*

\* Ввод информации текущего человека из потока ввода. Функция рассчитана на ввод с клавиатуры

\* @param in поток ввода. Так как функция рассчитана на ввод с клавиатуры, то в качестве данного параметра

\* передается переменная <code>cin</code>

\* @param people человек, в который занесется информация полученная с потока ввода

\* @return поток ввода для дальнейшего ввода данных

\*/

friend istream& operator >> (istream& in, People& people);

};

#endif //COURSEEXAMPLE\_PEOPLE\_H

9 Файл People.cpp. Реалізація класу People

/\*\*

\* @file People.cpp

\* Реализация класса жителя.

\*

\* @author davs

\* @version 0.0.1

\* @date 26-nov-2015

\*/

#include "People.h"

People::People() : fio(""), inn(0) { }

People::People(string newFio, long newInn) : fio(newFio), inn(newInn) { }

People::~People() { }

People::People(const People &newPeople) : fio(newPeople.fio), inn(newPeople.inn) { }

const string People::getFio() const {

return fio;

}

void People::setFio(const string &fio) {

People::fio = fio;

}

long People::getInn() const {

return inn;

}

void People::setInn(long inn) {

People::inn = inn;

}

bool operator<(const People &p1, const People &p2) {

return false;

}

bool operator>(const People &p1, const People &p2) {

return false;

}

ostream &operator<<(ostream &out, const People &people) {

return out;

}

istream &operator>>(istream &in, People &people) {

return in;

}

10 Файл SelfTester.h. Об’ява класа для самотестування розроблених методів

/\*\*

\* @file SelfTester.h

\* Объявление Класса для самотестирования разработанных методов.

\* @author davs

\* @version 0.0.1

\* @date 04-dec-2015

\*/

#ifndef COURSEEXAMPLE\_SELFTESTER\_H

#define COURSEEXAMPLE\_SELFTESTER\_H

/\*\*

\* Класс по самотестированию разработанных методов. В качестве эталонных (ожидаемых) результатов использовать данные,

\* полученные при помощи STL-базирующих классов / методов

\*/

class SelfTester {

private:

// Здесь должны присутствовать методы по валидации (assert) обектов, например

// \* метод, который сравнивает на эквивалетность двух студентов

// \* метод, сравнивающий размеры коллекций

public:

// Здесь должны присутствовать методы по тестированию разработанных функций в классах

// \* List

// \* ListProcessor

// \* ListWalker

// Каджый метод должен выполнять следующие действия:

// \* вывести на экран имя текущей функции (см. \_\_func\_\_ )

// \* сгенерировать данные для теста

// \* сравнить результаты с ожидаемыми (полученными при помощи STL классов)

// \* вывести результат теста, например OK (или Failed)

// Например,

// testInsert .................. OK

// testSaveToFileAndRestore .... FAILED [expected: [что ожидали получить] , actual: [что получили]]

};

#endif //COURSEEXAMPLE\_SELFTESTER\_H

11 Файл Student.h. Об’ява класу Student

/\*\*

\* @file Student.h

\* Объявление класса - студента.

\* @author davs

\* @version 0.0.1

\* @date 26-nov-2015

\*/

#ifndef COURSEEXAMPLE\_STUDENT\_H

#define COURSEEXAMPLE\_STUDENT\_H

#include "People.h"

/\*\*

\* Студент - Житель страны, помимо основных характеристик, обучающийся в ВУЗе

\*/

class Student : public People {

private:

string highSchoolName; ///< Имя ВУЗа

public:

/\*\*

\* Конструктор по-улолчанию. Инициализация полей пустыми начальными значениями.

\*/

Student();

/\*\*

\* Конструктор с параметрами. Инициализация характристик стужента переданными параметрами

\* @param newFio ФИО студента

\* @param newInn идентификационный номер (ИНН) студента

\* @param newHighSchoolName имя ВУЗа студента

\*/

Student(const string newFio, long newInn, const string newHighSchoolName);

/\*\*

\* Конструктор копирования. Инициализируем студента теми же характеристиками, что и переданный.

\* @param newStudent человек, по образу и подобию которого будем создавать текущего студента

\*/

Student(const Student& newStudent);

/\*\*

\* Деструктор. Очиска выделенной в конструкторе памяти.

\* В связи с тем, что в конструкторе ничего не выделялось, деструктор - пустой

\*/

virtual ~Student();

/\*\*

\* Получение имени ВУЗа, в которм учится студент

\* @return имя ВУЗа

\*/

const string getHighSchoolName() const;

/\*\*

\* Установка нового имени ВУЗа (в который, наверно, студент поступил)

\* @param highSchoolName имя ВУЗа

\*/

void setHighSchoolName(const string highSchoolName);

/\*\*

\* Вывод информации текущего студента в поток вывода. Функция рассчитана на вывод на экран, по этому имеет

\* следующий формат: '[ИНН (длиной 15 символов)]: [ВУЗ (до 15 символов)] : [FIO]'

\* @param out поток вывода. Так как функция рассчитана на вывод на консоль, то в качестве данного параметра

\* передается переменная <code>cout</code>

\* @param student студента, информация о котором передаем в поток вывода

\* @return поток вывода для дальнейшего вывода данных

\*/

friend ostream& operator << (ostream& out, const Student& student);

/\*\*

\* Ввод информации текущего студента из потока ввода. Функция рассчитана на ввод с клавиатуры

\* @param in поток ввода. Так как функция рассчитана на ввод с клавиатуры, то в качестве данного параметра

\* передается переменная <code>cin</code>

\* @param student студента, в который занесется информация полученная с потока ввода

\* @return поток ввода для дальнейшего ввода данных

\*/

friend istream& operator >> (istream& in, Student& student);

};

#endif //COURSEEXAMPLE\_STUDENT\_H

12 Файл Student.cpp. Реалізація класу Student

/\*\*

\* @file Student.cpp

\* Реализация класса студента.

\*

\* @author davs

\* @author xone

\* @version 0.0.1

\* @date 26-nov-2015

\*/

#include "Student.h"

Student::Student() : People(), highSchoolName("") { }

Student::Student(const string newFio, long newInn, const string newHighSchoolName)

: People(newFio, newInn),

highSchoolName(newHighSchoolName) { }

Student::Student(const Student &newStudent)

: People(newStudent.getFio(), newStudent.getInn()),

highSchoolName(newStudent.highSchoolName) { }

Student::~Student() { }

const string Student::getHighSchoolName() const {

return highSchoolName;

}

void Student::setHighSchoolName(const string highSchoolName) {

Student::highSchoolName = highSchoolName;

}

ostream &operator<<(ostream &out, const Student &student) {

return out;

}

istream &operator>>(istream &in, Student &student) {

return in;

}