Міністерство освіти і науки України

Сумський державний педагогічний університет імені А.С. Макаренка

Фізико-математичний факультет

Кафедра інформатики

**Марченко Владислав Володимирович**

**ДИПЛОМНА РОБОТА**

ВИВЧЕННЯ ПРОГРАМУВАННЯ В КУРСІ ІНФОРМАТИКИ ОСНОВНОЇ ШКОЛИ НА ПРИКЛАДІ СЕРЕДОВИЩА LAZARUS

Галузь знань: 0403 Системні науки та кібернетика

Напрям підготовки 6.040302 Інформатика\*

Кваліфікаційна робота на здобуття освітнього рівня «Бакалавр»

Науковий керівник:

старший викладач С.І. Петренко

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2018 року

Виконавець

студент 443 групи В.В. Марченко

ЗМІСТ

[УМОВНІ СКОРОЧЕННЯ 3](#_Toc516422780)

[ВСТУП 4](#_Toc516422781)

[РОЗДІЛ 1. МЕТОДИЧНІ ОСОБЛИВОСТІ НАВЧАННЯ УЧНІВ ПРОГРАМУВАННЮ У ШКІЛЬНОМУ КУРСІ ІНФОРМАТИКИ 7](#_Toc516422782)

[1.1. Місце теми «програмування» у навчальних програмах 7](#_Toc516422783)

[1.2. Мови програмування, що вивчаються в школі 11](#_Toc516422784)

[1.3. Середовище програмування Lazarus. Теоретичні відомості 20](#_Toc516422785)

[РОЗДІЛ 2. МЕТОДИЧНІ МАТЕРІАЛИ ДЛЯ НАВЧАННЯ УЧНІВ ТЕМИ «СЕРЕДОВИЩЕ ПРОГРАМУВАННЯ LAZARUS» 24](#_Toc516422786)

[2.1. Теоретична складова вивчення теми «Середовище програмування Lazarus» 24](#_Toc516422787)

[2.2. Практичні роботи з теми «середовище програмування lazarus» 49](#_Toc516422788)

[ВИСНОВКИ 59](#_Toc516422789)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 60](#_Toc516422790)

[ДОДАТКИ 63](#_Toc516422791)

# **УМОВНІ СКОРОЧЕННЯ**

ОС – операційна система.

ПЗ – програмне забезпечення.

IDE – інтегроване середовище розробки.

GNU – рекурсивний акронім **G**NU’s **N**ot **U**nix. Проект розробки вільного ПЗ.

GNU GPL – GNU General Public License. Ліцензія на безкоштовне використання та модифікацію ПЗ [1][1].

GNU LGPL – GNU Lesser General Public License. Ліцензія на безкоштовне використання та модифікацію ПЗ з можливістю використання в комерційних проектах [2][2].

LCL – Lazarus Component Library (бібліотека компонентів Lazarus) [3][3].

# **ВСТУП**

Розвиток творчих здібностей учнів із урахуванням їх пізнавальних інтересів є однією із основних задач сучасної української школи. Практична реалізація цієї проблеми покладена на ряд предметів, серед яких є і інформатика. За своїм потенціалом курс інформатики має розвивати в учнів аналітичне, синтетичне, логічне й критичне мислення, творчі здібності, естетичний смак, толерантність та повагу до чужого інтелектуального продукту, здатність аналізувати різноманітні процеси та явища й з’ясовувати їхні причинно-наслідкові та структурні зв’язки [4]. Інформатична освіта значною мірою впливає на вивчення всього комплексу математичних, природничих і технологічних предметів та дозволяє випускнику школи вирішувати широке коло практичних і теоретичних завдань у повсякденному житті.

Предметно-змістова лінія «Моделювання, алгоритмізація й програмування» є однією з основних у курсі вивчення інформатики в школі, оскільки вона є базовою для формування алгоритмічного способу мислення. У той же час, практика показує, що цей розділ є досить складним для засвоєння багатьом учням. Це обумовлює актуальність проблеми пошуку нових шляхів для вивчення цієї тематичної лінії у школі.

Навчальні програми з інформатики [4;5] дозволяють використовувати будь-які сучасні середовища об’єктно-орієнтованого візуального програмування. Кожне з цих середовищ має свої переваги та недоліки. Пакети Turbo Delphi 2006, Delphi 7 і Visual Basic є комерційними, а Visual Basic має вузьке застосування (лише під платформу Windows). Тому під час вивчення основ візуального програмування доречно використовувати платформу Lazarus, тому що вона є безкоштовною і розповсюджується на умовах GNU GPL та GNU LGPL в частині компонентів [6] і є зручним середовищем об’єктно-орієнтованого візуального програмування.

Основними функціональними характеристиками системи програмування Lazarus є:

* наявність потужного редактор коду із підсвічуванням синтаксису, системою підказок, гіпертекстовою навігацією по вихідних текстах, автозавершенням коду;
* реалізація набору основних елементів керування
* бібліотека візуальних компонентів LCL;
* наявність вбудованого відлагоджувача;
* наявність редактора форм та інспектора об’єктів;
* підтримка різних типів синтаксису Pascal;
* наявність власного формату керування пакетами;
* середовище функціонування – операційні системи Linux, Microsoft Windows, Mac OS X, FreeBSD, WinCE, OS/2;
* багатомовність інтерфейсу (включно з підтримкою української);
* створення машинного коду, який не потребує інтерпретатора і працює в десять разів швидше, ніж проміжний код;
* підтримка перетворення проектів Delphi.

Отже, комунікація в середовищі Lazarus має певні переваги:

* зручний для вивчення інтерфейс рідною мовою;
* при встановленні україномовного інтерфейсу не виникає проблем із розумінням інформації про наявність помилок.

Таким чином, Lazarus є досить перспективним засобом для вивчення програмування в школах

Об'єкт дослідження – навчання інформатики учнів основної школи.

Предмет дослідження – методика навчання учнів 8 класу з теми «Алгоритми та програми» на основі середовища Lazarus.

Метою роботи є дослідження можливостей використання системи Lazarus при вивченні шкільного курсу інформатики та курсу за вибором з основ візуального програмування.

Завдання:

1. Проаналізувати навчальні програми з інформатики для 5-9 класів
2. Розглянути доцільність вивчення мови програмування Lazarus
3. Розробити практичні завдання
4. Розробити методичні рекомендації до практичних завдань

# **РОЗДІЛ 1. МЕТОДИЧНІ ОСОБЛИВОСТІ НАВЧАННЯ УЧНІВ ПРОГРАМУВАННЮ У ШКІЛЬНОМУ КУРСІ ІНФОРМАТИКИ**

## **1.1. Місце теми «програмування» у навчальних програмах**

Мета української школи - виховати випускника, який володіє кількома іноземними мовами, має бажання і здатність до самоосвіти, здатний до підприємливості й ініціативності, має уявлення про світобудову, безпечно й доцільно використовує досягнення науки і техніки, дотримується здорового способу життя. До реалізації цієї мети покликані вносити вклад усі навчальні предмети, в тому числі, інформатика.

Згідно програм з інформатики, курс містить наступні навчальні лінії:

* інформація, інформаційні процеси, системи;
* комп’ютер як універсальний пристрій для опрацювання даних;
* комп’ютерні мережі;
* інформаційні технології створення та опрацювання текстових документів, графічних зображень, числових даних, об’єктів мультимедіа, комп'ютерних презентацій;
* комп’ютерне моделювання;
* основи алгоритмізації та програмування.

Українські школи в роботі використовують два варіанти навчальної програми з інформатики для 5-9 класів [7]. Перший варіант програми для учнів, що не вивчали «Сходинки до інформатики» в 2-4 класах, позначимо, як «п.1», а варіант програми для учнів, що вивчали «Сходинки до інформатики» в 2-4 класах – як «п.2». В обох варіантах приділяється досить велика увага саме темам алгоритмізації та програмування. Так, тематична лінія «Алгоритмізація та програмування» має вивчатися не менше 30% навчального часу, але не більше 40% [4].

Аналіз вивчення тематичної лінії «Алгоритмізація та програмування» по класах, у відповідності до програм п.1 та п.2, зроблений нами і представлений в таблиці 1.

Таблиця 1

Вивчення тематичної лінії алгоритмізації та  
програмування в 5-9 класах

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Клас  Тема | **5** | | **6** | | **7** | | **8** | | **9** | |
| п.1 | п.2 | п.1 | п.2 | п.1 | п.2 | п.1 | п.2 | п.1 | п.2 |
| Поняття алгоритму |  | + | + |  |  |  |  |  |  |  |
| Алгоритмічні структури |  | + | + |  | + |  |  |  |  |  |
| Складання алгоритмів |  | + | + |  |  |  |  |  |  |  |
| Вибір алгоритмічної структури для розв’язування задачі |  | + |  |  |  |  |  |  |  |  |
| Вкладені алгоритмічні структури |  |  |  | + |  |  |  |  |  |  |
| Структура програми |  |  |  |  |  |  |  | + |  |  |
| Сучасні мови програмування та їх елементи |  |  |  |  |  |  |  | + |  |  |
| Поняття об’єкту та його властивостей |  |  |  | + |  |  | + |  |  |  |
| Методи об’єкта |  |  |  |  |  |  | + | + |  |  |
| Поняття події |  |  |  | + |  |  | + |  |  |  |
| Поняття величини, змінної, операції присвоєння |  |  |  |  |  | + | + |  |  |  |
| Використання змінних у алгоритмах |  |  |  |  |  | + |  |  |  |  |
| Поняття константи |  |  |  |  |  |  |  | + |  |  |
| Одновимірні масиви |  |  |  |  |  |  |  |  | + | + |
| Методи опрацювання елементів масиву |  |  |  |  |  |  |  |  | + | + |

Тематична лінія «Алгоритмізація та програмування» включає теми: «Поняття алгоритму», «Алгоритмічні структури», «Складання алгоритмів», «Вибір алгоритмічної структури для розв’язування задачі», «Вкладені алгоритмічні структури», «Структура програми», «Сучасні мови програмування та їх елементи», «Поняття об’єкту та його властивостей», «Методи об’єкта», «Поняття події», «Поняття величини, змінної, операції присвоєння», «Використання змінних у алгоритмах», «Поняття константи», «Одновимірні масиви», «Методи опрацювання елементів масиву». Теми «Поняття алгоритму», «Алгоритмічні структури» та «Складання алгоритмів» у варіанті п.2 вивчаються в 5 класі, а у п.1 – в 6 класі. Це три перших теми всієї тематичної лінії. У п.2 в 5 класі також передбачено вивчення теми «Вибір алгоритмічної структури для розв’язування задачі». У 6 класі в п.2 вивчаються теми «Вкладені алгоритмічні структури», «Поняття об’єкту та його властивостей» та «Поняття події». У 7 класі в п.1 вивчається лише одна тема – «Алгоритмічні структури», а в п.2 – «Поняття величини, змінної, операції присвоєння» та «Використання змінних у алгоритмах». У 8 класі п.1 включає всі теми, що стосуються об’єктів, тобто теми », «Поняття об’єкту та його властивостей», «Методи об’єкта», «Поняття події» та «Поняття величини, змінної, операції присвоєння», а п.2 – «Структура програми», «Сучасні мови програмування та їх елементи», «Поняття константи» та «Методи об’єкта». На завершення, в 9 класі в обох варіантах навчальних програм вивчаються теми, пов’язані з масивами даних – «Одновимірні масиви» та «Методи опрацювання елементів масиву».

Можна зробити висновок, що програма для учнів, що вивчали інформатику в 2-4 класах, є більш розширеною за своєю структурою. Тематична лінія алгоритмізації та програмування починається в 5 класі, а в програмі для учнів, що не вивчали інформатику раніше – в 6 класі. З таблиці 1 видно, що програма для учнів, що вивчали інформатику в 2-4 класах є більш широкою, проте гірше структурованою. Для підтвердження цього висновку приведемо приклади:

* властивості об’єкта вивчаються в 6 класі, а методи – аж у 8;
* змінні вивчаються у 7 класі, а константи – у 8;
* у 8 класі вивчаються дві більш загальні теми («Структура програми», «Сучасні мови програмування та їх елементи»), одна тема, що визначає базове поняття («Поняття константи»), та одна тема, що стосується об’єктів («Методи об’єкта»)

Незважаючи на це, обидві програми дозволяють в достатній мірі опрацювати матеріал з тематичної лінії алгоритмізації та програмування.

## **1.2. Мови програмування, що вивчаються в школі**

**Scratch**

Середовище програмування Scratch дозволяє дітям створювати власні анімовані та інтерактивні проекти: комікси, ігри, анімаційні фільми та інші твори. Ними можна обмінюватися всередині міжнародного середовища, яке поступово формується в мережі Інтернет.

В основі Scratch лежать традиції мови програмування Лого. У середовищі Scratch користувач з окремих цеглинок (блоків програми) збирає свій мультимедійний проект подібно до конструкції з кубиків Лего. Проста форма дозволяє навіть самим маленьким долучатися до програмування, перетворюючи навчання в захоплюючу гру. Але, почавши з гри, можна далі розвивати і розширювати своє вміння будувати і програмувати. Середовище програмування Scratch створювалося спеціально для того, щоб підлітки 8 - 16 років використовували його самостійно. Це нове технологічне середовище дозволяє їм виразити себе в комп’ютерній творчості.

У шкільному курсі інформатики при розгляді вступу до алгоритмізації та програмування рекомендується використовувати навчальне середовище програмування Scratch [8;9]. В результаті використання простих команд можна скласти складну модель, в якій будуть взаємодіяти об’єкти, наділені різними властивостями. Це візуальне середовище програмування, розроблене спеціально для навчання алгоритмізації, об’єктно-орієнтованого підходу в програмуванні та подієвих складових. На офіційному сайті Scratch позиціонується як мова програмування, призначена для навчання. Саме середовище написане на мові програмування Squeak, а віртуальна машина, в якій відбуваеться трансляція коду та його виконання – на мові Node.js (принаймні, починаючи з версії 3.0) [10;11]. Scratch розроблений у 2006р. під керівництвом Мітчела Рєзніка групою Lifelong Kindergarten в лабораторії Media Lab Массачусетського технологічного інституту. Scratch працює на комп’ютерах з операційними системами Windows, MacOS та Linux. На даний час розроблено дві версії програмного середовища: Scratch 1.4, і Scratch 2.0. Принципи роботи в них схожі. Друга версія вирізняється більш сучасним і зручним інтерфейсом, можливістю працювати online в мережі Інтернет, використовувати веб-камеру та деякі інші можливості. Водночас версія 1.4 містить усе необхідне для роботи і не вимагає підключення до мережі Інтернет, вона більше адаптована до україномовного користувача, тому розпочинати знайомство з середовищем Scratch доцільніше з неї. Версія 2.0 дає можливість розробляти, редагувати і переглядати проекти безпосередньо в браузері, але може працювати автономно.

Повністю візуалізований спосіб створення невеликих програм виявився досить ефективним засобом навчання, саме тому Scratch почав використовуватися в таких вищих навчальних закладах світу, як Гарвардський університет, Массачусетський технологічний інститут (більш відомий як MIT) та багатьма менш відомими закладами світу [10;12;13].

Серед переваг цього середовища можна виділити наочність створення та роботи програм, низький «поріг входження» – створення програм дається відносно легко як учням з гуманітарним нахилом, так і учням з добре розвиненими математичними здібностями. До недоліків можна віднести відсутність можливостей створення прикладних програм, що працюють незалежно від середовища та відсутність можливостей роботи на відносно низькому рівні (пряма взаємодія з файлами та пам’яттю, графічною підсистемою ОС та ін.). Середовище доцільно використовувати для вивчення тематичної лінії «Алгоритмізація та програмування» для дітей до 12 років.

**Visual Basic**

Вже досить довгий час у школі вивчається мова програмування Visual Basic. Вона є рекомендованою до вивчення в деяких підручниках [14], незважаючи на те, що мова досить застаріла, та має ряд проблем.

Microsoft Visual Basic – це засіб розробки програмного забезпечення, створений в 1991р. і підтримуваний корпорацією Microsoft, який складається з мови програмування і середовища розробки. Мова програмування Visual Basic є діалектом мови програмування Basic, та частково успадкувала її синтаксис. У той же час, мова програмування Visual Basic поєднує в собі процедури, елементи об'єктно-орієнтованих та компонентно-орієнтованих мов програмування. Середовище розробки містить інструменти для візуального конструювання користувацького інтерфейсу.

Мова програмування Visual Basic вважається потужним засобом швидкої розробки прототипів програм, програм, що працюють з базами даних, і взагалі для компонентного способу створення програм, що працюють під управлінням майже усіх версій операційних систем сімейства Microsoft Windows [15;16]. Ця мова позиціонується як об’єктно-орієнтована, але в ній немає механізму успадкування класів. Таким чином, необхідно явно записувати всі методи. Крім цього, в мові програмування Visual Basic відсутні вказівники, низькорівневий доступ до пам’яті та ASM-включення. Також, розроблена програма вимагає встановлених бібліотек з динамічним зв’язуванням, що досить часто стає предметом критики. Мова програмування Visual Basic не є крос-платформною мовою програмування.

У середині 1998 року вийшла остання версія цієї мови – Visual Basic 6.0. Після цього Microsoft різко змінила політику стосовно мов сімейства Basic. Замість розвитку мови програмування Visual Basic було створено абсолютно новий програмний продукт – Visual Basic.NET, перша версія якого з’явилася в 2001. Це принципово нова мова, що має, окрім синтаксису, дуже мало схожого з Visual Basic 6.0 та відрізняється від нього так сильно, як у свій час Visual Basic відрізнявся від QBASIC, на основі якого був створений. Visual Basic.NET відразу ж зайняв місце Visual Basic в Visual Studio і на цьому розвиток класичного Visual Basic припинився [15].

Мова програмування Visual Basic – досить потужний засіб розробки програмного забезпечення, що є скомпільованим, проте потребує деяких динамічно зв’язуваних бібліотек. Середовище має елементи візуального програмування, а сама мова є об’єкто-орієнтованою. Також, мова програмування Visual Basic має досить строгий синтаксис. Все це дозволяє використовувати дану мову програмування при вивченні програмування в школі, незважаючи на те, що вона вже є досить застарілою.

**Pascal**

Це одна з найбільш відомих мов програмування [17], використовується для навчання програмуванню в старших класах [18] і на перших курсах вишів, є базою для ряду інших мов.

Мова Pascal була створена Ніклаусом Віртом в 1968-1969 роках після його участі в роботі комітету розробки стандарту мови Алгол-68. Мова названа в честь французького математика, фізика, літератора і філософа Блеза Паскаля, який створив першу в світі механічну машину, що вміла додавати два числа. Перша публікація Вірта про мову датована 1970 роком; представляючи цю мову, автор в якості мети її створення вказував побудову невеликої та водночас ефективної мови, що сприяє хорошому стилю програмування, та використовує структурне програмування і структуровані дані.

Особливостями мови є строга типізація та наявність засобів структурного (процедурного) програмування. Мова Pascal була однією з перших таких мов. На думку Вірта, мова повинна сприяти дисциплінованому програмуванню, тому, разом зі строгою типізацією, в мові Pascal зведені до мінімуму можливі синтаксичні неоднозначності, а сам синтаксис автор зробив інтуїтивно зрозумілим навіть при першому знайомстві з мовою.

Проте, спочатку мова мала ряд обмежень: неможливість передачі в функції масивів змінної довжини, відсутність нормальних засобів роботи з динамічною пам'яттю, обмежена бібліотека введення-виведення, відсутність можливості підключення функцій, що написані на інших мовах програмування, відсутність інструментів роздільної компіляції та ін. Деякі недоліки Паскаля були виправлені в ISO-стандарті 1982 року, зокрема, в мові з'явилися відкриті масиви, що дали можливість використовувати одні і ті ж процедури для обробки одновимірних масивів різних розмірів.

Однак деякі недоліки мови не проявляються або навіть стають перевагами при навчанні програмуванню. До 1980-х років Паскаль став основою для створення численних навчальних програм, в окремих випадках на його основі було створено спеціалізовані навчальні мови програмування, так, на початку 1980-х років в СРСР для навчання школярів основам інформатики та обчислювальної техніки Андрій Єршов розробив паскалеподібну «навчальну алгоритмічну мову».

Найбільш відомою реалізацією мови програмування Pascal, що забезпечила широке поширення і розвиток мови, є мова Turbo Pascal фірми Borland, що переросла потім в об'єктний Паскаль (Object Pascal) для ОС DOS (починаючи з версії 5.5) і Windows, і далі в Delphi, в якій були впроваджені значні розширення мови.

Мова програмування Pascal є досить потужним засобом розробки компільованого програмного забезпечення. Для неї створено кілька середовищ, проте жодне з них не має елементів візуального програмування, її синтаксис та логіка досить прості, що дозволяє використовувати її при вивченні програмування в школі. Сама мова програмування вважається застарілою, проте на її базі створено низку інших мов та середовищ, що на сьогодні є досить популярними.

**C++**

Це статично типізована мова програмування загального призначення. Програми, написані на цій мові потребують попередньої компіляції перед виконанням. Вона підтримує такі парадигми програмування, як процедурне програмування, об'єктно-орієнтоване програмування, узагальнене програмування. Ця мова має багату стандартну бібліотеку, яка включає в себе поширені контейнери і алгоритми введення-виведення, регулярні вирази, підтримку багатопоточності та інші можливості. Мова програмування C++ поєднує властивості як високорівневих, так і низькорівневих мов [19;20]. У порівнянні з його попередником – мовою програмування C, більшу увагу приділено підтримці об'єктно-орієнтованого і узагальненого програмування [20].

Мова програмування C++, будучи однією з найпопулярніших, широко використовується для розробки програмного забезпечення. Область її застосування включає створення операційних систем, різноманітних прикладних програм, драйверів пристроїв, додатків для вбудованих систем, високопродуктивних серверів та розважальних програм. Існує досить велика кількість реалізацій мови програмування C++, і безкоштовних, і комерційних, і для різних платформ. Наприклад, на платформі x86 це GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder і інші. Мова програмування C++ значним чином вплинула на інші мови програмування, в першу чергу на Java та C#.

Синтаксис мови програмування C++ успадкований від мови програмування C. Одним з принципів розробки було збереження сумісності з C. Проте, C++ не є в строгому сенсі надмножиною C – кількість програм, які можуть однаково успішно оброблятися як компіляторами C, так і компіляторами C ++, досить велика, але не включає всі можливі програми на C.

Доступ до можливостей стандартної бібліотеки мови C++ забезпечується за допомогою включення в програму (за допомогою директиви #include) відповідних стандартних заголовкових файлів. Всього в стандарті C++11 визначено 79 таких файлів. Засоби стандартної бібліотеки оголошуються в просторі імен std. Заголовки, імена яких відповідають шаблону «cX», де X - ім'я заголовки стандартної бібліотеки C без розширення (cstdlib, cstring, cstdio та ін.), містять оголошення, відповідні даної частини стандартної бібліотеки мови C. Стандартні функції бібліотеки мови C також знаходяться в просторі імен std.

Мова програмування C++ – дуже потужний засіб розробки компільованого програмного забезпечення різного рівня складності. Для цієї мови створено досить багато середовищ, що спеціалізовані на розробці програмного забезпечення для певної платформи – від мікроконтроллерів до високонавантажених серверів. Не рекомендується для вивчення в школі, адже є досить складною.

**Python**

Одна з мов програмування, що рекомендована до вивчення в школі [21]. Для Python розроблено досить багато середовищ (IDE), проте їх основною проблемою є відсутність (або недостатня кількість) готових компонентів, що потребує від учня переходу від складання алгоритмів до написання коду одразу. Наприклад, в підручнику для 8 класу авторського колективу під керівництвом Н.В. Морзе розглядається одне з найвідоміших та найчастіше використовуваних середовищ – PyCharm [21]. Дане середовище взагалі не має вбудованих готових компонентів. В свою чергу, це стимулює вивчення окремих особливостей мови замість складання алгоритмів. Але школа не повинна випускати програмістів з окремої мови програмування. Окрім всього цього, мова програмування Python не є C-подібною, а отже учню, що захоче в подальшому вивчати програмування на інших мовах буде необхідно звикати до абсолютно іншого синтаксису.

Python – мова програмування, що активно розвивається, нові версії (з додаванням та зміною мовних властивостей) виходять приблизно один раз в два з половиною роки. Внаслідок цього і деяких інших причин для мови програмування Python відсутні стандарти ANSI, ISO та інші офіційні стандарти, їх роль виконує CPython.

Мова програмування Python портована і працює майже на всіх відомих платформах – від КПК до мейнфреймів. Існують порти під Microsoft Windows, практично всі варіанти UNIX (включаючи FreeBSD і Linux), Plan 9, Mac OS і Mac OS X, iPhone OS 2.0 і вище, Palm OS, OS/2, Amiga, HaikuOS, AS/400 і навіть OS/390, Windows Mobile, Symbian і Android [22].

Мова програмування Python – засіб розробки інтерпретованого програмного забезпечення різного рівня складності. Програми, що написані за допомогою мови програмування Python вимагають встановленої відповідної версії інтерпретатора на стороні користувача. Для цієї мови створено досить багато середовищ. Рекомендується до вивчення у 8-9 класах основної школи, зокрема, у підручниках за редакцією груп авторів під керівництвом Н.В. Морзе та Й.Я. Ривкінда.

**Lazarus**

Проект Lazarus почав розроблятися ще у 1999 році програмістами Кліффом Баесманом, Шейном Міллером та Майклом А. Гессом на базі проекту Megido. Приблизно до вересня 2000 р., Маттіас Гертнер зробив вагомий вклад у розвиток програми, зокрема її інтерфейсу. Він же через три роки додав пакетну систему і продовжував додавати додаткові функції та вдосконалення IDE.

Міха Нелісен почав робити свій внесок у червні 2003 року, переважно відправляючи виправлення для інтерфейсу win32. У той час Lazarus був заснований на GTK для win32 і Linux. На Linux він працював дуже добре, але на win32 було досить багато помилок в реалізації. Користувачі Lazarus все частіше запитували про рідний інтерфейс win32, і Міха взяв реалізацію в свої руки.

З 2005 року почали розроблятися нові інтерфейси (Qt, WinCE, fpGUI та Cocoa), а також SVN-база. Зараз Lazarus є готовим продуктом, проте створення нових елементів та бібліотек не припиняється.

Дане середовище має добре структуровану візуалізовану структуру, що дозволяє створювати деякі прості програми лише за допомогою миші. Таким чином, в процесі створення програми, учень може візуально оцінювати кінцевий результат. Середовище складається з кількох вікон, що тісно зв’язані між собою. Так, змінюючи властивості певного поміщеного на форму об’єкта, що мають візуальний еквівалент, останній зміниться і на формі.

До того ж, це середовище, найчастіше рекомендоване до вивчення офіційними джерелами [21;23;24]. Виходячи з сукупності всього вищесказаного, вважаю доцільним вивчення в школі саме середовища програмування Lazarus. В офіційній реалізації, на момент написання наукової роботи, має 243 готові компоненти, що дозволяє без проблем створювати прості програми майже без написання коду [25-27]. З іншого боку, для створення більш складних алгоритмів необхідно розуміти принципи об’єктно-орієнтованого програмування та вміти записувати алгоритми за допомогою мови програмування. Існує навіть набір компонентів для створення власних. Існує також клас для створення багатопотокової програми - *TThread* [28]. Виходячи з цього всього, можна стверджувати, що за допомогою середовища Lazarus можна створювати програми будь-якого рівня складності. Окрім цього, Lazarus написаний на базі крос-платформного компілятора FPC, що дає змогу створювати програми для більшості сучасних платформ [29]. Більшість його компонентів розповсюджуются на умовах ліцензії GNU LGPL (Lesser General Public License), що дає змогу використовувати їх у комерційних проектах [6].

**1.3. Середовище програмування Lazarus. Теоретичні відомості**

В поясненні нового матеріалу можуть використовуватися наступні поняття:

* Вікно – неперервна область екрану, на якій відображається інформація, що є результатом роботи програми; заголовок та кнопки керування програмою та є незалежною від інших областей
* Контрол – елемент вікна програми, з яким може взаємодіяти користувач
* Діалог – вікно, що використовуються безпосередньо для взаємодії з користувачем
* Клік – натискання клавішою миші
* HID (Human Interface Devices) – клас приладів, що безпосередньо взаємодіють з користувачем
* IDE (Integrated Development Environment) – комплексне програмне рішення для розробки програмного забезпечення

Всі елементи середовища, про які йтиме мова надалі були протестовані та перевірені в наступних версіях програми:

* Mac OS X 10.12 Sierra: версія 1.8.2 (2018-04-19), платформа i386-darwin-qt4 [26]
* Windows 7 SP1: версія 1.8.2 (2017-04-14), платформа x86\_64-win32-qt4 [27]
* ArchLinux Manjaro KDE Edition 17.1.8: версія 1.8.0 (2017-12-23) та 1.8.2 (2018-04-14), платформа x86\_64-linux-qt5 [25]

**Запуск програми**

Відразу після першого запуску програми з’являється вікно налаштувань (див. [додаток А](#Додаток_А)). Тут присутні декілька вкладок. Вкладка «Lazarus» відповідає за налаштування головної папки; вкладка «Compiler» – за файл запуску компілятора (можна використовувати компілятор для специфічної платформи); вкладка «FPC Source» – за папку з кодом FPC; вкладка «Make» – за файл запуску програми-збірника; вкладка «Debugger» – за файл запуску програми-налагоджувача. За замовчуванням всі ці вкладки налаштовані правильно для тієї платформи, на якій запускається IDE. Зміна значень в цих вкладках необхідна лише тоді, коли необхідно використати компілятор для іншої платформи, розширений налагоджувач, змінені компоненти та інше. Для запуску основного інтерфейсу необхідно натиснути «Start IDE».

**Основний інтерфейс** (див. [додаток Б](#Додаток_Б))

Основний інтерфейс складається з кількох вікон, що відповідають за різні функції середовища. Ліве вертикальне вікно відповідає за параметри виділеного об'єкта. Верхнє горизонтальне – за панель інструментів (відкриття/збереження/запуск проекту, тощо) та панель об'єктів, де можна обрати необхідний елемент та помістити його на форму, що за замовчуванням показана на передньому плані та називається Form1. Відразу за нею знаходиться редактор коду, де саме й пишеться логіка роботи програми.

Опис головних функціональних вікон

**Верхнє вікно (панель інструментів та панель об’єктів)** (див. [Додаток В](#Додаток_В)) відповідає за набір елементів форми (видимих та невидимих).  
Воно має 15 вкладок різного призначення.

* Standard (елементи форми, що використовуються найчастіше)
* Additional (додаткові елементи форми)
* Common Controls (контроли, що призначені для більш структурованого відображення інформації)
* Dialogs (вікна, що використовуються для обробки специфічних даних, що передані користувачем)
* Data Controls (елементи, що використовуються для зберігання невеликих об’ємів специфічних даних)
* Data Access (елементи, що використовуються для доступу до даних, збережених в пам’яті)
* System (елементи, що відповідають за низькорівневі системні компоненти)
* SQLdb (елементи, що відповідають за взаємодію з базами даних SQL)
* Misc (елементи, що повністю не задовольняють жодної класифікації)
* SynEdit (елементи форми, призначені для редагування коду програм на різних мовах програмування)
* LazControls (контроли, реалізації яких немає в ObjectPascal, специфічні контроли, написані для Lazarus)
* RTTI (контроли, написані спеціальним чином для того, щоб можна було напряму з’єднуватися з опублікованими властивостями цих компонентів)
* IPro (низькорівневі провайдери IP)
* Chart (елементи, пов’язані з побудовою/виведенням графіків)
* Pascal Script (елементи для інтеграції власного коду як звичайного елементу; необхідні для реалізації власних компонентів)

**Панель властивостей (інспектор об’єктів)** (див. [додаток Г](#Додаток_Г))

Вікно зліва відповідає за набір властивостей обраного елемента. Воно має всього 4 вкладки.

* Properties. Відповідає за властивості елемента
* Events. Відповідає за призначення функцій подіям цього елемента
* Favorites. В цій вкладкі розміщуються ті властивості та події, що використовуються найчастіше
* Restricts. В цій вкладкі вказані обмеження елемента відносно платформ

**Панель редагування коду (редактор коду)**

Центральне вікно відповідає за логіку роботи програми. Саме в ньому записується так званий «текст програми». Воно має вбудовані функції автодоповнення та підсвічування синтаксису та тісно зв’язане з іншими вікнами середовища.

**Форма**

Найменше вікно відповідає за візуальне відображення створеної програми на екрані. Воно показує попередній вигляд програми та також дуже тісно зв’язане з іншими вікнами середовища. Обравши елемент на формі, ми зможемо відредагувати його властивості на відповідній панелі. Подвійним кліком лівої клавіші миші ми зможемо потрапити в панель редагування коду, а курсор опиниться відразу на місці обробника події OnClick відповідного об’єкту. Для додавання об’єкту, досить обрати його на панелі об’єктів та помістити його на форму кліком лівої клавіші миші на обраному місці.

# **РОЗДІЛ 2. МЕТОДИЧНІ МАТЕРІАЛИ ДЛЯ НАВЧАННЯ УЧНІВ ТЕМИ «СЕРЕДОВИЩЕ ПРОГРАМУВАННЯ LAZARUS»**

## **2.1. Теоретична складова вивчення теми «Середовище програмування Lazarus»**

Lazarus - це надмножина Pascal. Це означає, що програма, написана на мові Pascal є правильно написаною програмою на Lazarus, проте програма Lazarus не обов'язково є правильно написаною на Pascal.

Pascal – мова структурного програмування. Це означає, що програміст повинен висловлювати свої думки дуже дисципліновано, з використанням малого числа чітко обумовлених конструкцій, використовуючи як чергування їх, так і вкладення один в одного. Конструкції бувають лінійні, розгалуджені та циклічні.

**Лінійні конструкції.** Лінійними конструкціями називають звичайну послідовність дій. Реалізація послідовності дій (тобто структури слідування) виконується за допомогою складеного оператора: begin <послідовність операторів> end. Службові слова begin і end часто називають операторними дужками.

begin

<оператор 1>;

<оператор 2>;

...

<оператор n>

end;

Не рекомендується (хоча і можна) використовувати оператор безумовного переходу goto.

begin

start: *// куди будемо повертатися*

writeln(1);

goto start; *// програма ніколи не завершиться*

end;

**Конструкції розгалудження.** Для реалізації розгалуджень в Pascal передбачені два оператора: умовний оператор і оператор варіанту (вибору). Вони призначені для виділення зі складових їх операторів одного, який і виконується.

Структура і дія умовного оператора такі:

if <логічний вираз> then

<оператор 1>

else

<оператор 2>

Умовний оператор може бути неповним, тобто не містити частину «else <оператор 2>». В цьому випадку, якщо значення логічного виразу false, умовний оператор не викликає ніяких дій.

program cross;

var a, b, c, d, m, n, l:real;

begin

writeln('Введіть координати кінців відрізків');

read(a,b,с,d);

writeln;

if a<c then m:=c else m:=a;

if b<d then n:=b else n:=d;

if m<n then l:=n-m else l:=0;

writeln('Довжина спільної частини відрізків = ', l)

end.

Існують також складені умови (ті, які складаються з декількох умов). Для їх створення необхідно використовувати логічні оператори. Серед них є бінарні та один унарний оператор. Унарним називається оператор, який застосовується до однієї умови, а бінарним – до двох. Унарним є оператор not, який істинний тоді, і тільки тоді, коли сама умова хибна. Бінарних операторів три: and (істинний тоді і тільки тоді, коли обидві умови істинні одночасно), or (істинний тільки тоді, коли хоча б одна з умов істинна) та xor (істинний тоді і тільки тоді, коли лише одна умова істинна).

if (a > b) or not (a > c) then ...

Умовні оператори також можуть бути вкладені. Наприклад:

if cond1 then begin

...

if cond2 then begin

...

end;

...

end;

Оператор варіанту (case) має наступну форму:

case <вираз> of

<список констант 1>: <оператор 1>;

<список констант 2>: <оператор 2>;

...

<список констант N>: <оператор N>;

else <оператор за замовчуванням>;

end

Вираз, що стоїть між службовими словами case і of, повинен мати значення ординального (0, 1, 2, …) або рядкового типу (останній додається за рахунок компіляції коду як Object Pascal за рахунок додавання рядку {$mode objfpc}{$H+}). Будь-який список констант може складатися з однієї константи. Оператор варіанту обчислює значення виразу, записаного після case. Якщо його значення збігається з однією з констант в заданому списку, то виконується оператор, що стоїть після цього елементу списку. Якщо значення виразу не співпало з жодною константою, то програма очікує оператор else, а якщо нема і останнього, то нічого не робить.

program seasons;

var

k: integer

begin

writeln('введіть номер місяця');

readin(k);

case k of

1, 2, 12: writeln('зима');

3, 4, 5: writeln('весна');

6, 7, 8: writeln('літо');

9, 10, 11: writeln('осінь')

end;

end.

**Конструкції повтору (цикли).** Для реалізації циклів в Pascal є три оператора. Якщо кількість повторень відомо заздалегідь, то зручно скористатися оператором циклу з параметром. В інших випадках слід використовувати оператори циклу з передумовою (цикл «поки») або з післяумовою (цикл «до»).

Цикл з передумовою є найбільш потужним. Інші оператори циклу можна виразити через нього. Форма його запису така: while <логічний вираз> do <оператор>

program store;

var

sum, w: real;

num: integer;

begin

num := 0;

sum := 0;

while sum < 100 do

begin

writeln('введіть вагу грузу машини');

readln(w);

sum := sum + w;

if sum <= 100 then

num := num + l

else

writeln('груз вже не поміститься')

end;

writeln('число розвантажених машин = ', num);

end.

Оператор циклу з післяумовою має наступну форму запису: repeat <послідовність операторів> until <логічний вираз>

program store2;

var

sum, w: real;

man: integer;

begin

num := 0; sum := 0;

repeat

writeln('введіть вагу грузу машини');

readln(w); sum:=sum+w;

if sum<=100 then

num := num+l

else

writeln('груз вже не поміститься')

until sum >= 100;

writeln('кількість розвантажених машин =', num);

end.

Оператор циклу з параметром передбачає повторне виконання деякого оператора з одночасною зміною за правилом арифметичної прогресії значення керуючої змінної (параметра) цього циклу. Оператор циклу з параметром має дві форми.

Форма1: for <параметр>: = <вираження1> to <вираз 2> do <оператор>

Параметр, вираз 1, вираз 2 повинні бути одного ординальне типу; Параметр в цьому циклі зростає.

Якщо в цьому описі відношення <= замінити на> =, а функцію succ на pred, то параметр в циклі буде спадати, в цьому випадку цикл з параметром приймає форму 2.

Форма 2: for <параметр>: = <вираз 1> downto <вираз 2> do <оператор>

program mili;

const a = 1.603; b = 'км'; с = 'милі';

var

k: integer; m: real;

begin

writeln(b, c);

writeln;

for k:=l to 10 do

begin

m:=k/a;

writeln(k, m)

end

end.

Запишемо в цій програмі цикл з параметром в формі 2:

for k := 10 downto 1 do

begin

m := k/a;

writeln(k, m)

end

**Функції та процедури.** Процедури і функції узагальнено називають підпрограмами. Вони являють собою автономні блоки інструкцій, які можуть бути викликані з різних місць програми. Функція – це підпрограма, яка при виконанні повертає значення. Процедура – це підпрограма, яка не повертає значення.

Виклики функцій, оскільки вони повертають значення, можуть бути використані як вирази в інструкціях присвоєння та інших операціях. Наступний код демонструє цю властивість:

I := SomeFunction(X);

Цей запис викликає SomeFunction і привласнює результат змінній I. Виклик функції не може виконуватися в лівій частині інструкції присвоювання.

Виклики процедур (і функцій при включеному розширеного синтаксису {$X+}), можуть являти собою завершення інструкції. Наприклад, запис DoSomething; викликає підпрограму DoSomething в тому випадку, якщо DoSomething є функцією; Значення, що повертається – відкидається.

Процедури і функції можуть рекурсивно викликати самі себе.

Коли ви оголошуєте процедуру або функцію, ви вказуєте її ім'я, кількість і тип параметрів, які вона приймає, і, в тому випадку, якщо ви оголошуєте функцію, вам необхідно вказати тип значення, яке вона повертає (ця частина оголошення іноді називається прототипом або заголовком). Потім ви пишете блок програмного коду, який виконується при виклику процедури або функції. Ця частина іноді називається тілом або блоком підпрограми.

Оголошення процедури має наступний синтаксис:

procedure **procedureName**(parameterList);

localDeclarations;

begin

statements

end;

procedureName – будь-який допустимий ідентифікатор, statements – це послідовність інструкцій, яка виконується при виклику процедури, (parameterList) і localDeclarations; – список параметрів та зона локальних оголошень, вони є необов’язковими. Нижче наведено приклад оголошення процедури:

procedure **NumString**(N: Integer; var S: string); directives;

var

V: Integer;

begin

V := Abs(N);

S := ;

repeat

S := Chr(V mod 10 + Ord('0')) + S;

V := V div 10;

until V = 0;

if N < 0 then

S := '-' + S;

end;

Після цього оголошення ви можете викликати процедуру NumString наступним чином:

NumString(17, MyString);

Виклик цієї процедури привласнює значення 17 змінній MyString (яка повинна бути рядкового типу).

Усередині блоку інструкцій процедури ви можете користуватися змінними та іншими ідентифікаторами, оголошеними в розділі localDeclarations. Крім того, ви можете використовувати імена параметрів зі списку параметрів (як, наприклад, N і S в наведеному прикладі). Список параметрів визначає безліч локальних змінних. Не слід намагатися повторно оголосити в секції localDeclarations змінні з іменами, зазначеними в списку параметрів. Нарешті, ви можете використовувати будь-які ідентифікатори, в області видимості яких оголошена процедура.

Оголошення функції схоже з оголошенням процедури і відрізняється тим, що в ньому вказується тип значення і, безпосередньо, повертається значення. Оголошення функції має вигляд

function **functionName**(parameterList): returnType; directives;

localDeclarations;

begin

statements

end;

У даному оголошенні functionName – це будь-який допустимий ідентифікатор, returnType – ідентифікатор типу, statements – послідовність інструкцій, яка виконується при виконанні функції, а (parameterList), directives; і localDeclarations; є необов’язковими.

У тілі функції діють ті ж правила, які відносяться і до процедур. Усередині блоку інструкцій ви можете користуватися ідентифікаторами, оголошеними в секції localDeclarations, параметрами, зазначеними в parameterList і будь-якими ідентифікаторами, в області видимості яких оголошена функція. Крім того, ім’я функції так само діє як спеціальна змінна, яка містить значення, що повертається функцією. Аналогічно діє і змінна Result.

При включенні розширеного синтаксису ({$X+}), змінна Result неявно оголошується в кожної функції. Не намагайтеся оголосити її повторно. приклад:

function **WF**: Integer;

begin

WF := 17;

end;

Цей запис оголошує функцію з ім’ям WF, яка не має параметрів і завжди повертає цілочисельне значення 17. Це оголошення еквівалентно наступному:

function **WF**: Integer;

begin

Result := 17;

end;

Далі приведений приклад більш складної функції:

function **Max**(A: array of Real; N: Integer): Real;

var

X: Real;

I: Integer;

begin

X := A[0];

for I := 1 to N - 1 do

if X < A[I] then X := A[I];

Max := X;

end;

У тілі функції ви можете повторно задавати значення змінної Result або імені функції (значення, що задаються повинні бути відповідного типу). Коли виконання функції припиняється, останнє значення, присвоєне змінній Result або імені функції, стає значенням, що повертається функцією. наприклад:

function **Power**(X: Real; Y: Integer): Real;

var

I: Integer;

begin

Result := 1.0;

I := Y;

while I > 0 do

begin

if Odd(I) then Result := Result \* X;

I := I div 2;

X := Sqr(X);

end;

end;

Result і ім’я функції завжди мають однакове значення. Для прикладу, наступна функція поверне значення 11:

function **MyFunction**: Integer;

begin

MyFunction := 5;

Result := Result \* 2;

MyFunction := Result + 1;

end;

Незважаючи на це, змінна Result і ім'я функції не повністю взаємозамінні. Коли ім'я функції виявляється в лівій частині інструкції привласнення, компілятор припускає, що його необхідно використовувати як змінну Result, та коли ім'я функції виявляється в будь-якому іншому місці в тілі функції – компілятор інтерпретує його як рекурсивний виклик. Змінна Result може бути використана як змінна в інструкціях, перетвореннях типів, конструкторах множин, індексах і викликах інших підпрограм.

Якщо при виконанні функції, її виконання переривається до того моменту, коли змінної Result або імені функції буде присвоєно якесь значення, функція поверне невизначене значення.

Досить часто виникає ситуація, коли необхідно написати «універсальну» функцію, що буде підтримувати більшу кількість типів вхідних даних. Такі функції оголошуються декілька разів з директивою overload та називаються «перевантаженими». Наприклад, можна написати функцію ділення, що при вхідних типах данних числа з плаваючою комою повертає результат того ж типу, а при вхідних цілих – повертає цілу частину від ділення

function **Divide**(X, Y: Real): Real; overload;

begin

Result := X/Y;

end

function **Divide**(X, Y: Integer): Integer; overload;

begin

Result := X div Y;

end;

Ці оголошення створюють дві функції з ім’ям Divide, які приймають параметри різних типів. Коли ви викликаєте Divide, компілятор за списком переданих підпрограмі параметрів визначає, яку функцію потрібно підключити. Наприклад Divide(6.0, 3.0) викликає першу функцію Divide, оскільки її аргументами є числами з плаваючою комою.

Ви можете передати перевантаженій підпрограмі параметри, що не співпадають за типом з параметрами, зазначеними у всіх оголошеннях цієї підпрограми і при цьому сумісні за типом більш ніж з одним оголошенням. Найчастіше таке відбувається при оголошенні перевантажених підпрограм, які приймають в якості параметрів різні цілі або дійсні типи.

procedure **Store**(X: Longint); overload;

procedure **Store**(X: Shortint); overload;

У випадках, якщо можливо уникнути невизначеності, компілятор підключає підпрограму, параметри якої мають найменший діапазон, що задовольняє параметрам виклику (слід пам’ятати, що дійсні константи завжди мають тип Extended).

Перевантажені підпрограми повинні відрізнятися за кількістю прийнятих параметрів або за їх типом. Таким чином, наступна пара оголошень викличе помилку компіляції:

function **Cap**(S: string): string; overload;

...

procedure **Cap**(var Str: string); overload;

...

При цьому наступна пара оголошень є вірною:

function **Func**(X: Real; Y: Integer): Real; overload;

...

function **Func**(X: Integer; Y: Real): Real; overload;

...

Компілятор здатний відрізняти функції, які беруть в якості параметрів значення типів AnsiString/PAnsiChar, UnicodeString/PChar і WideString/PWideChar на одних і тих же позиціях. Строкові або символьні константи, що передаються в перевантажені функції, транслюються в первинний рядковий або символьний тип, тобто в UnicodeString/PChar.

procedure **test**(const A: AnsiString); overload;

procedure **test**(const W: WideString); overload;

procedure **test**(const U: UnicodeString); overload;

procedure **test**(const PW: PWideChar); overload;

var

a: AnsiString;

b: WideString;

c: UnicodeString;

d: PWideChar;

e: string;

begin

a := 'a';

b := 'b';

c := 'c';

d := 'd';

e := 'e';

test(a); *// calls AnsiString version*

test(b); *// calls WideString version*

test(c); *// calls UnicodeString version*

test(d); *// calls PWideChar version*

test(e); *// calls UnicodeString version*

test('abc'); *// calls UnicodeString version*

test(AnsiString('abc')); *// calls AnsiString version*

test(WideString('abc')); *// calls WideString version*

test(PWideChar('PWideChar')); *// calls PWideChar version*

end;

Значення типу variant також можуть бути передані в якості параметрів для перевантажених функцій. На відміну від простих типів, variant обробляється як більш узагальнений тип. Перевага завжди віддається точному збігу типів. Якщо значення типу variant передається в перевантажену підпрограму, яка приймає значення variant на цій позиції параметра, це розглядається як точний збіг для типу variant.

Це може стати причиною деяких побічних ефектів у випадку з дійсними типами. Числа з плаваючою точкою порівнюються за розміром. Якщо для дійсної змінної, переданої як параметр перевантаженій підпрограмі, не знаходиться точного збігу, але присутній параметр типу variant, він приймається за дійсний тип з меншим діапазоном.

procedure **foo**(i: integer); overload;

procedure **foo**(d: double); overload;

procedure **foo**(v: variant); overload;

var

v: variant;

begin

foo(1); *// версия с integer*

foo(v); *// версия с variant*

foo(1.2); *// версия с variant*

*//( константа с плавающей точкой -> тип extended)*

end;

Цей приклад викликає версії процедури foo з параметром variant, а не double, оскільки константа 1.2 явно визначається як тип extended, який не є точним збігом для типу double. Extended так само не є точним збігом для variant, але variant розглядається як більш узагальнений тип (в той час як double є типом з діапазоном, меншою ніж у extended).

Виклик з перетворенням типу (foo(Double(1.2));) не буде працювати. Замість цього слід використовувати типізовані константи:

const d: double = 1.2;

begin

foo(d);

end;

Наведений вище код працює коректно і викликає версію foo з параметром double. Наступний приклад також викликає версію foo з параметром double. Single краще збігається за типом з double, ніж з variant.

const s: single = 1.2;

begin

foo(s);

end;

При оголошенні декількох перевантажених підпрограм, найкращим способом уникнути обробки дійсного типу як variant є оголошення окремої версії підпрограми для кожного дійсного типу (Single, Double, Extended) крім версії з типом variant.

Якщо використовувати параметри за замовчуванням в перевантажених процедурах, необхідно уникати оголошень невизначених параметрів.

Щоб уникнути можливих побічних ефектів при перевантаженні, необхідно специфікувати ім'я підпрограми при виклику. Наприклад, Unit1.MyProcedure(X, Y) може викликати тільки підпрограми, оголошені в Unit1; якщо в модулі Unit1 не опиниться підпрограм, список параметрів яких співпаде з дійсними параметрами, виникне помилка.

Тіло функції або процедури часто починається з оголошень локальних змінних, використовуваних в блоці інструкцій підпрограми. Ці оголошення можуть так само включати константи, типи і інші підпрограми. Область видимості локальних ідентифікаторів обмежується областю оголошення підпрограми.

Функції та процедури всередині своїх блоків локальних оголошень іноді можуть містити інші функції і процедури. Наприклад, наступне оголошення процедури з ім'ям DoSomething містить оголошення вкладеної процедури.

procedure **DoSomething**(S: string);

var

X, Y: Integer;

procedure **NestedProc**(S: string);

begin

...

end;

begin

...

NestedProc(S);

...

end;

**Клас.** Клас або тип класу визначає структуру, що складається з полів, методів і властивостей. Примірники класів називаються об’єктами. Іншими словами, класи – це те, з чого створюються об’єкти. У кожного класу є власні поля, властивості та методи.

Поле являє собою змінну, яка є частиною об’єкта. Як і поля записів, поля класів представляють елементи даних, які присутні в кожному примірнику класу.

Метод – це процедура або функція, пов’язана з класом. Велика частина методів оперує об'єктами (тобто екземплярами класів). Деякі методи (звані методами класу) працюють з класами як з типами.

Властивість – це інтерфейс до даних, пов’язаних з об’єктом (які часто зберігаються в полях). Властивості мають так званий специфікатор доступу, який визначає, яким чином дані можуть бути прочитані або змінені. З інших частин програми, тобто зовні об'єкта, властивість в більшості випадків виглядає як поле.

Об’єкти – це блоки пам’яті, що виділяються динамічно, структура яких визначається їх типом. Кожен об'єкт має унікальну копію полів, визначених у описі класу, але всі екземпляри класів використовують однакові методи. Об'єкти створюються і руйнуються за допомогою спеціальних методів, які називаються конструкторами і деструкторами.

Змінна типу клас насправді є вказівником, який посилається на об’єкт. Як і інші вказівники, змінні типу клас можуть приймати значення nil. Немає необхідності явно розіменовувати змінну типу клас для того, щоб отримати доступ до об'єкта, на який вона посилається. Тобто, операцію SomeClass.Size := 100; немає необхідності записувати як SomeClass^.Size := 100;

Клас повинен бути оголошений раніше, ніж буде створено його примірник (не можна оголосити клас всередині оголошення змінної). Оголошувати клас слід тільки в зовнішній області видимості програми або модуля, але не всередині оголошення процедури або функції. Оголошення класу має вигляд:

type

className = class [abstract | sealed] (ancestorClass)

memberList

end;

У цьому разі className – будь-який допустимий ідентифікатор;

*sealed* і *abstract* - опціональні зарезервовані слова; Тут синтаксис [*abstract* | *sealed*] позначає, що може бути використано тільки одне із зарезервованих слів. При цьому дужки і символ "|" не повинні бути присутніми в оголошенні.

Якщо клас позначений ключовим словом sealed, успадкування від цього класу забороняється.

Цілий клас може бути оголошений абстрактним навіть в тому випадку, якщо він не містить в своєму описі абстрактних віртуальних методів. Абстрактними називаються ті методи, які позначені для переоголошення в классах, що наслідуються з данного.

Слід зазначити, що для забезпечення сумісності, Lazarus дозволяє створювати екземпляри класу, оголошеного як abstract, але користуватися такою можливістю не слід.

Клас не може бути одночасно оголошений як abstract і sealed.

(AncestorClass) - Вказівка на клас-предок, може не вказуватися; якщо пропущено вказівку на клас-предок (ancestorClass), новий клас успадковується безпосередньо від класу System.TObject. Якщо в оголошенні класу включається вказівка на клас-предок і список компонентів класу порожній, в оголошенні можна не вказувати ключового слова end. Оголошення класу може так само включати список інтерфейсів, що реалізуються класом.

memberList – оголошення компонентів класу (поля, властивості і методи);

Методи описуються в оголошенні класу у вигляді заголовків, без вказівки тіл самих методів. Програмний код методів повинен бути зазначений в інших частинах програми. Далі наведено оголошення класу TMemoryStream з модуля Classes:

type

TMemoryStream = class(TCustomMemoryStream)

private

FCapacity: Longint;

procedure **SetCapacity**(NewCapacity: Longint);

protected

function **Realloc**(var NewCapacity: Longint): Pointer;

virtual;

property **Capacity**: Longint read FCapacity

write SetCapacity;

public

destructor **Destroy**; override;

procedure **Clear**;

procedure **LoadFromStream**(Stream: TStream);

procedure **LoadFromFile**(const FileName: string);

procedure **SetSize**(NewSize: Longint); override;

function **Write**(const Buffer; Count: Longint): Longint;

override;

end;

Клас Classes.TMemoryStream походить від Classes.TCustomMemoryStream, наслідуючи його методи. При цьому він перевизначає кілька методів і властивостей, включаючи деструктор. Конструктор Create успадковується без змін від класу System.TObject (завдяки чому він не вказується в оголошенні). Кожен компонент класу, оголошується як private, protected, public (в оголошенні немає методів, оголошених як published). Значення цих ключових слів пояснюється нижче.

Після оголошення класу можна створити його екземпляр:

var stream: TMemoryStream;

stream := TMemoryStream.Create;

При оголошенні класу ви можете вказати його безпосереднього попередника. Наприклад, наступне оголошення type TSomeControl = class(TControl); оголошує клас з ім'ям TSomeControl, успадкований від Vcl.Controls.TControl. Клас автоматично успадковує всі компоненти класу-попередника. Кожен клас може оголосити нові компоненти і перевизначити успадковані, але не може видаляти компоненти, визначені в класі-попереднику. Отже TSomeControl містить всі компоненти, визначених у Vcl.Controls.TControl і його попередниках.

Область видимості ідентифікатора компонента починається від його оголошення, завершується кінцем оголошення класу і розширюється за рахунок всіх спадкоємців класу і блоків коду методів, визначених в класі і всіх його спадкоємців.

Клас System.TObject, оголошений в модулі System, є обов'язковим предком всіх інших класів. System.TObject визначає лише кілька методів, включаючи конструктор і деструктор. До того ж до System.TObject, модуль System оголошує посилання на тип класу System.TClass:

TClass = class of TObject;

Якщо при оголошенні класу опускається попередник, клас успадковується безпосередньо від System.TObject.

type

TMyClass = class

...

end;

Таке оголошення буде еквівалентним наступному:

type

TMyClass = class(TObject)

...

end;

Тип класу сумісний за присвоєнням з попередниками. З цього випливає, що змінна типу клас може посилатися на екземпляр будь-якого класу попередника.

type

TFigure = class(TObject);

TRectangle = class(TFigure);

TSquare = class(TRectangle);

var

Fig: TFigure;

Змінній Fig можна привласнити значення типу TFigure, TRectangle або TSquare. Всередині функцій та процедур може існувати ситуація, коли буде необхідно отримати властивість об’єкта TSquare, переданого як аргумент типу TFigure. У такому разі спочатку у нас немає доступу до властивостей потрібного об’єкта, а тому необхідно явно вказати тип. Для цього існує ключове слово as. Принцип його застосування досить простий:

procedure **proc**(obj: TFigure)

var

myRect: TRectangle;

begin

myRect = obj as TRectangle;

...

end

Кожен компонент класу має атрибут, званий видимістю, який визначається одним із зарезервованих слів private, protected, public або published. Наприклад:

published property Color: TColor read GetColor write SetColor;

В коді вище, оголошується властивість Color з областю видимості published та специфікаторами читання та запису. Видимість визначає, де і як буде доступний компонент класу. Найменша доступність у видимості private, protected має середню доступність, а видимість public, published і automated мають найбільшу доступність.

Якщо в оголошенні компонента класу опускається специфікатор видимості, компонент має таку ж видимість, як у компонента, оголошеного перед ним. Компоненти без специфікаторів видимості на початку оголошення класу за замовчуванням отримують видимість published (якщо у компілятора встановлений режим {$M+}, або якщо клас, від якого успадковується оголошений клас, скомпільовано в режимі {$M+}) або public (у всіх решті випадків).

Для зручності читання коду, найкращим рішенням буде організувати оголошення класу з урахуванням видимості його компонентів: в початок оголошення поміщати компоненти з видимістю private, потім – компоненти з видимістю protected і так далі. В цьому випадку зарезервоване слово для позначення видимості вказується одноразово на початку "секції". Таким чином, типове оголошення класу виглядає так:

type

TMyClass = class(TControl)

private

*{ private declarations here }*

protected

*{ protected declarations here }*

public

*{ public declarations here }*

published

*{ published declarations here }*

end;

Перевизначаючи клас-спадкоємець, ви можете збільшити видимість його компонентів, але не знизити її. Наприклад, властивість, позначена як protected в класі-предкі, може бути оголошено як public в класі-спадкоємці, але не може бути оголошено як private. Більш того, компоненти з видимістю published не можуть отримати видимість public в класі-спадкоємці.

Компоненти з видимістю private невидимі зовні модуля або програми, в якій оголошено клас. Іншими словами, метод з видимістю private не може бути викликаний з іншого модуля, а властивість з такою видимістю не може бути прочитано або змінено з іншого модуля.

Розміщуючи пов'язані оголошення класів в одному і тому ж модулі, ви можете надати класам доступ до чужих компонентів з видимістю private, але при цьому все таки обмежуєте доступ до них.

Компоненти з видимістю protected доступні в модулі, де оголошений клас, а також в будь-якому класі-спадкоємці (незалежно від того, в якому модулі він оголошений). Метод класу-попередника з видимістю protected може бути викликаний з оголошення будь-якого методу, що належить класу-спадкоємцю, і будь-яку властивість (або поле) з видимістю protected може бути прочитано або змінено в методі класу-спадкоємця. Компоненти, призначені для використання при реалізації класів-спадкоємців зазвичай мають видимість protected.

Компоненти з видимістю public видимі всюди, де доступний їх клас.

Компоненти з видимістю published мають таку ж видимість, як і компоненти, помічені як public. Різниця між ними полягає в тому, що інформація про типи в режимі виконання програми (run-time type information - RTTI) створюється для компонентів з видимістю published. RTTI дозволяє додатку динамічно отримувати інформацію про поля, властивості та методи класу. RTTI використовується для отримання доступу до значень властивостей при завантаженні і збереженні файлів, відображення властивостей об'єкта в інспекторі об'єктів і зв'язку спеціальних методів (обробників подій) з властивостями-подіями.

Компоненти з видимістю published мають обмеження за типами даних. Компоненти можуть бути порядкових, строкових, інтерфейсних, варіантних типів, типу клас і покажчиками на метод. Крім того, "опублікованим" може бути будь-який дійсний тип (крім Real48). Властивості типу масив (array type), на відміну від властивостей-масивів (array properties), не можуть мати видимість published.

Деякі властивості хоч і можуть мати видимість published, вони не повністю підтримуються потоковою системою. Це відноситься до типів запис, властивостей-масивів та властивостей перелічуваних типів, що мають анонімні значення. Якщо ви встановлюєте видимість published для таких властивостей, інспектор об’єктів не буде коректно відображати їх, а значення властивості не буде зберігатися при збереженні об'єктів на диск.

Всі методи можуть бути опубліковані, але клас не може публікувати більше одного перезавантаженого (overfowed) методу одночасно.

Клас не може мати компонентів з видимістю published за винятком тих випадків, коли він компілюється з ключем компілятора {$M+} або успадковується від класу, скомпільованої в режимі {$M+}. Більшість класів, що мають компоненти з видимістю published, успадковуються Classes.TPersistent, який скомпільовано з ключем {$M+}. Таким чином, необхідність в частому використанні директиви $M немає.

*Зауваження*: В секціях published класів заборонено розміщення ідентифікаторів, що містять символи Unicode. Крім того, ідентифікатори, які містять символи Unicode заборонені при оголошенні типів, які використовують члени з видимістю published.

## **2.2. Практичні роботи з теми «середовище програмування lazarus»**

Результати всіх завдань, представлених надалі перевірені на працездатність в автоматичному режимі та опубліковані у відкритому доступі за умовами ліцензії GNU GPL v3.0 та вище на сайті https://github.com (див. [додаток Д](#Додаток_Д))

**Практична робота № 1**

**Тема: Знайомство з середовищем програмування Lazarus**

*Мета*: Ознайомитися з функціональними елементами середовища Lazarus та їх функціями (вікнами, контролами та діалогами).

*Завдання*: Необхідно створити просту віконну програму, на якій буде лише одна кнопка, при натисканні на яку відкривається діалогове вікно з текстом.

*Хід роботи*:

1. Відкрийте середовище програмування Lazarus.
2. На панелі об’єктів оберіть звичайну кнопку (TButton).
3. Помістіть кнопку на форму.
4. Подвійним кліком лівої кнопки миші на створеній кнопці перейдіть до панелі редагування коду.
5. Всередині конструкції *begin – end* впишіть наступний код ShowMessage('Привіт від Lazarus!');.
6. Запустіть програму на виконання (кнопкою ⏵).
7. Натисніть кнопку всередині вікна, що з’явилось.

*Завдання для самостійної роботи*:

1. Створити форму з кнопкою, що виводить повідомлення з Вашим іменем та прізвищем.
2. На панелі властивостей обрати зелений колір для кнопки, що виводить повідомлення «У мене все вийшло».

**Практична робота № 2**

**Тема: Об’єктна складова середовища програмування Lazarus**

*Мета*: Ознайомитись з подіями та властивостями об’єктів в середовищі програмування Lazarus.

*Завдання*: Створити програму, що змінює колір форми при наведенні курсору на відповідну кнопку.

*Хід роботи*:

1. Відкрийте середовище Lazarus.
2. Створіть дві звичайні кнопки (TButton), розмістивши їх на формі.
3. Найменуйте кнопки як «Зелений» та «Червоний» відповідно (властивість Caption).
4. Створіть нові обробники подій *OnMouseEnter* для кнопок.
5. В обробниках подій змініть властивість *Color* об’єкта *Form1* на *clGreen* та *clRed* відповідно.
6. Збережіть проект у власній папці.
7. Запустіть програму на виконання (кнопкою ⏵).
8. Наводьте курсор миші над кнопками почергово та оцініть результат роботи програми.

*Завдання для самостійної роботи*:

1. Створити програму, що змінює колір форми при наведенні курсору на відповідну кнопку (в двох кольорах: білий та синій).
2. Створити програму, що змінює колір форми при наведенні курсору на відповідну кнопку, а при відведенні – на стандартний.

**Практична робота № 3**

**Тема: Умовний оператор if**

*Мета*: Ознайомитись з умовним оператором if.

*Завдання*: Оптимізувати раніше створену програму, що змінює колір форми при наведенні курсору на відповідну кнопку.

*Хід роботи*:

1. Відкрийте середовище Lazarus.
2. Відкрийте проект практичної роботи 2 за допомогою меню «Файл».
3. Створіть нову процедуру під назвою «OnMouseEnterButton» та оголосіть її в класі TForm1.
4. За допомогою умовного оператору *if* перевірте умову співпадіння параметру Sender з об’єктом Button1. У разі співпадіння виконайте зміну властивості Color об’єкту Form1 на clGreen. У разі неспівпадіння – на clRed.
5. Призначте обробником події OnMouseEnter обох кнопок процедуру OnMouseEnterButton.
6. Видаліть процедури, що більше не використовуються.
7. Збережіть проект.
8. Запустіть програму на виконання (кнопкою ⏵).
9. Наводьте курсор миші над кнопками почергово та оцініть результат роботи програми .

*Завдання для самостійної роботи*:

1. Доповнити створену програму ще однією кнопкою «Жовтий» та продублювати поведінку інших кнопок, створивши ще один умовний блок else if.
2. У раніше створеній програмі виконати перевірку співпадіння не об’єктів, а властивості Caption переданого об’єкту з назвою кольору українською мовою. Для отримання доступу до властивості Caption необхідно за допомогою оператора as представити об’єкт як клас TButton. Назва кольору записується як рядкова величина.

**Практична робота № 4**

**Тема: Умовний оператор case**

*Мета*: Ознайомитись з умовним оператором case.

*Завдання*: Оптимізувати раніше створену програму з завдання 3 з додаванням кількох кнопок.

*Хід роботи*:

1. Відкрийте середовище Lazarus.
2. Відкрийте проект з завдання 3 за допомогою меню «Файл» .
3. Додайте дві кнопки до форми та змініть їх властивості Caption на «Синій» та «Звичайний» .
4. За допомогою умовного оператору *case* перевірте умову співпадіння властивості Caption параметру Sender з рядками Зелений, Червоний та Синій. Для доступу до властивості Caption явно задайте тип переданого об’єкту як TButton (case (Sender as TButton).Caption). У разі співпадіння змініть властивість Color об’єкту Form1 на clGreen, clRed та clBlue відповідно. У разі неспівпадіння з хоча б одним із них – на clDefault.
5. Збережіть проект.
6. Запустіть програму на виконання (кнопкою ⏵).
7. Наводьте курсор миші над кнопками почергово та оцініть результат роботи програми.
8. Спробуйте переписати умови за допомогою оператору if та оцініть переваги використання оператору case для перевірки кількох варіантів умов.

*Завдання для самостійної роботи*:

1. У створену програму додати ще одну кнопку, що буде відповідати за рожевий колір.
2. Створити нову програму, що має одне поле для введення, кнопку, та за допомогою оператора case зрівнює введене значення з заданими. При введенні числа 4, програма повинна показати повідомлення «Влучив», 3 або 5 – «Майже влучив», а інші – «Ні, ну це зовсім далеко. Спробуй ще раз» .

**Практична робота № 5**

**Типи даних. Функції перетворення типів даних**

*Мета*: Ознайомитись з основними контролами, типами даних, функціями їх зміни та арифметичними операціями.

*Завдання*: Створити програму для підрахунку суми двох чисел, розміщених в різних полях введення.

*Хід роботи*:

1. Відкрийте середовище Lazarus.
2. Створіть новий проект за допомогою меню «Файл» .
3. Додайте два поля введення (TEdit) до форми.
4. Змініть властивості Text обох полів, видаливши стандартний текст.
5. Додайте до форми звичайну кнопку (TButton) .
6. Змініть властивість Caption створеної кнопки на «Рахувати» .
7. Додайте до форми текстове поле (TLabel) .
8. Змініть властивість Caption доданого поля, видаливши стандартний текст.
9. Створіть обробник події OnClick для кнопки.
10. Перед початком процедури оголосіть три змінних (first, second та sum) типу Extended.
11. Після ключового слова begin надайте цим змінним значень StrToFloat(Edit1.Text), StrToFloat(Edit2.Text) та first + second відповідно.
12. Одразу після надання значень змінним, виведіть результат на екран, надавши властивості Caption об’єкта Label1 значення  
    'Результат: ' + FloatToStr(sum) .
13. Збережіть проект.
14. Запустіть програму на виконання (кнопкою ⏵).
15. Введіть два довільних числа в поля введення та натисніть «Рахувати» .

*Завдання для самостійної роботи*:

1. У створеній програмі змініть ім’я змінної на sub, а її знічення – на різницю першого та другого числа.
2. Створити програму, що з двох чисел, введених в різні поля введення, обирає найбільше.

**Практична робота № 6**

**Тема: Користувацькі функції**

*Мета*: Ознайомитись з функціями, створити власну функцію.

*Завдання*: Модернізувати програму, створену в завданні 5 так, щоб існувала функція, що рахує суму чисел, переданих у вигляді рядкової змінної, та повертала результат у тому ж форматі.

*Хід роботи*:

1. Відкрийте середовище Lazarus.
2. Відкрийте проект з завдання 5 за допомогою меню «Файл» .
3. Створіть функцію FloatSum, що приймає два аргументи (float1 та float2) типу String та повертає дані такого ж типу.
4. Скопіюйте оголошення змінних в функцію.
5. Надайте змінним наступних значень: StrToFloat(float1), StrToFloat(float2) та first + second відповідно
6. Поверніть значення FloatToStr(sum) .
7. Видаліть оголошення змінних та їх означення в процедурі (обробнику події) .
8. В обробнику події надайте властивості Caption об’єкта Label1 значення 'Результат: ' + FloatSum(Edit1.Text, Edit2.Text).
9. Запустіть програму на виконання (кнопкою ⏵).
10. Перевірте роботу програми.

*Завдання для самостійної роботи*:

1. Створити програму, що має дві кнопки, після натиснення на які викликається функція, що рахує квадрат переданого числа. Після цього, результат виводиться у вікні повідомленні на екран для числел 7 та 8 відповідно.
2. Створити програму, що рахує заданий степінь заданого числа після натиснення на кнопку (число та степінь вводяться у різних полях введення) .

**Практична робота № 7**

*Мета*: Ознайомитись з можливостями інтеграції інших мов програмування в Lazarus.

*Завдання*: Створити віконну програму з двома полями введення для цілих чисел та кнопкою для їх взаємного ділення націло за допомогою скриптової мови Lua 5.3.

*Хід роботи*:

1. Відкрийте середовище Lazarus.
2. Додайте два поля введення (TEdit) до форми.
3. Змініть властивості Text обох полів, видаливши стандартний текст.
4. Додайте до форми звичайну кнопку (TButton) .
5. Змініть властивість Caption створеної кнопки на «Поділити націло» .
6. Створіть обробник події OnClick для кнопки.
7. Перед початком процедури оголосіть цілочисельну змінну res.
8. Після початку процедури надайте змінній res значення результату роботи функції luadivide з параметрами StrToInt(Edit1.Text) та StrToInt(Edit2.Text).
9. Створіть функцію luadivide наступним чином:

function **luadivide**(x, y: integer): integer;

var

L: plua\_state; *(\* змінна для збереження об'єкта Lua \*)*

script: string;

begin

*(\* задаємо ім’я файлу \*)*

script := 'divide.lua';

*(\* оголошуємо новий стан об'єкту Lua \*)*

L := luaL\_newstate();

*(\* відкриваємо необхідні бібліотеки \*)*

luaL\_openlibs(L);

*(\* відкриваємо та виконуємо скрипт \*)*

if luaL\_dofile(L, pChar(script)) <> 0 then

begin

*(\* показуємо помилку, якщо вона виникла \*)*

ShowMessage('Виникла помилка ' + lua\_tostring(L, -1));

end;

*(\* ім'я функції, яку ми використовуємо \*)*

lua\_getglobal(L, 'divide');

*(\* перший аргумент, який ми передаємо в функцію \*)*

lua\_pushinteger(L, x);

*(\* другий аргумент, який ми передаємо в функцію \*)*

lua\_pushinteger(L, y);

*(\* викликаємо функцію з 2 аргументами, повертаємо 1 результат \*)*

lua\_call(L, 2, 1);

*(\* отримуємо результат \*)*

luadivide := lua\_tointeger(L, -1);

*(\* очищуємо пам'ять \*)*

lua\_pop(L, 1);

*(\* закриваємо об'єкт \*)*

lua\_close(L);

end;

1. Збережіть проект в окремій папці.
2. Завантажте файл lua53.pas, що необхідний для роботи з цією мовою з ресурсу https://goo.gl/eLBY1o та збережіть в папці з проектом.
3. Завантажте бібліотеки для роботи для роботи з мовою з ресурсу https://sourceforge.net/projects/luabinaries/ та розмістіть їх в папці з проектом (Windows та Mac OS), чи встановіть їх за допомогою стандартного менеджера пакетів (Linux) .
4. Створіть файл divide.lua з наступним вмістом:

function **divide**(a, b)

return a // b

end

1. Запустіть програму на виконання (кнопкою ⏵).
2. Перевірте роботу програми.
3. Не закриваючи вікно програми, змініть // на + в файлі divide.lua. Натисніть кнопку «Поділити націло» знову.
4. Зробіть висновки, чи потрібна перекомпіляція основної програми для зміни логіки роботи скриптових файлів.

*Завдання для самостійної роботи*:

1. У створеній програмі додати кнопку, що викликає функцію mysecondfunc з двома аргументами, що задана в файлі main.lua. Тіло функції скопіювати з функції divide файлу divide.lua.
2. Створити функцію join для «склеювання» двох рядкових величин. Функцію записати в файлі main.lua. Показати результат роботи функції скрипта у вікні основної програми для введених користувачем рядків.

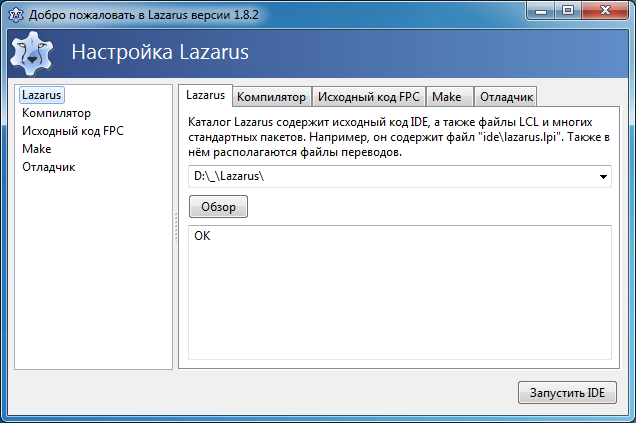
# **ВИСНОВКИ**

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

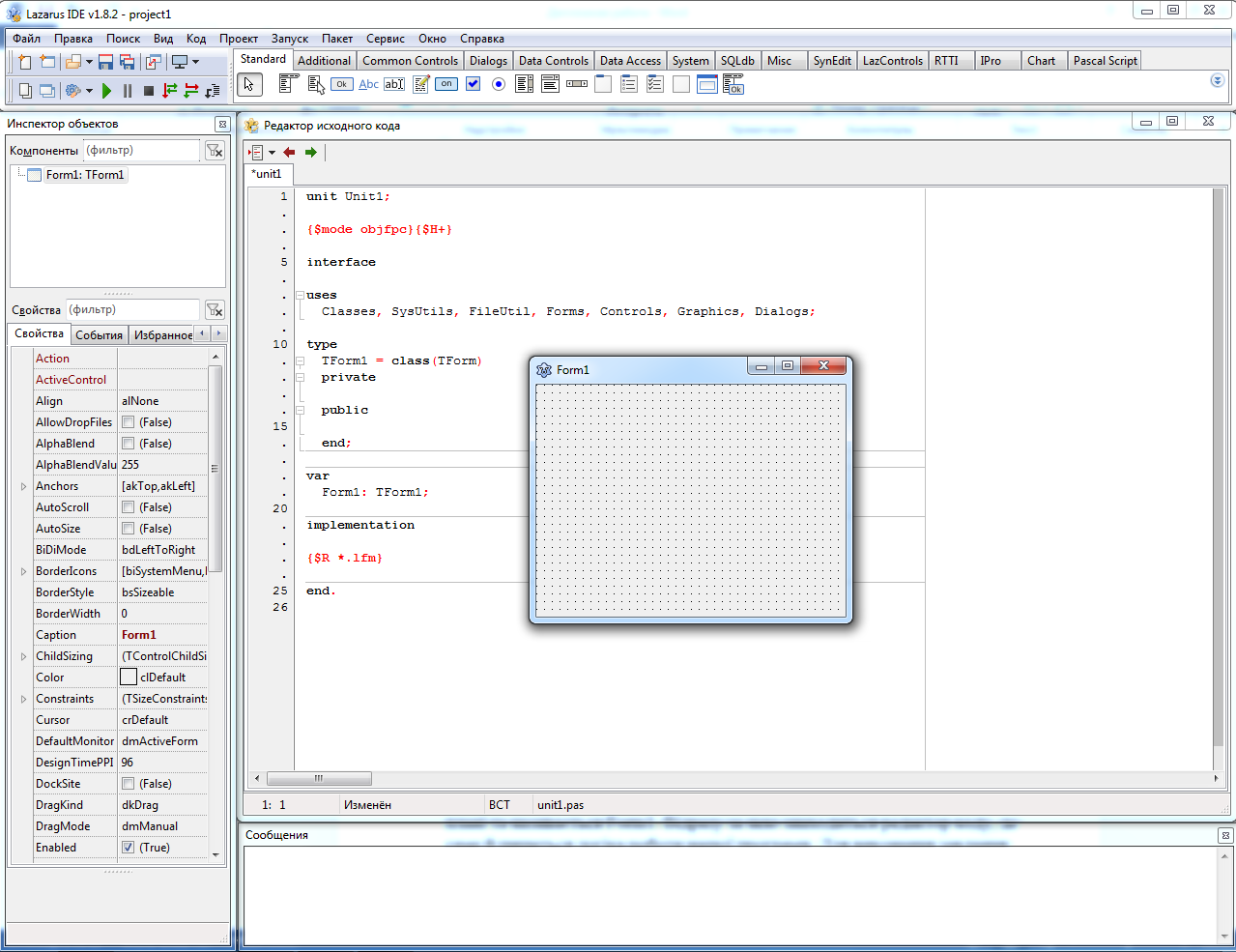
1. GNU General Public License [Електронний ресурс] – Режим доступу до ресурсу: http://www.gnu.org/licenses/gpl.en.html
2. GNU Lesser General Public License [Електронний ресурс] – Режим доступу до ресурсу: http://www.gnu.org/licenses/lgpl.en.html
3. GitHub Lazarus repository / COPYING.txt [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/graemeg/lazarus/blob/upstream/COPYING.txt
4. Інформатика. Навчальна програма для учнів 5-9 класів, які вивчали інформатику у 2-4 класах загальноосвітніх навчальних закладів [Електронний ресурс] – Режим доступу до ресурсу: https://mon.gov.ua/storage/app/media/zagalna%20serednya/programy-5-9-klas/onovlennya-12-2017/8-informatika.docx
5. Інформатика. Навчальна програма для учнів 5-9 класів загальноосвітніх навчальних закладів [Електронний ресурс] – Режим доступу до ресурсу: https://mon.gov.ua/storage/app/media/zagalna%20serednya/programy-5-9-klas/onovlennya-12-2017/programa-informatika-5-9-traven-2015.pdf
6. Lazarus, Wiki. licensing [Електронний ресурс] – Режим доступу до ресурсу: http://wiki.lazarus.freepascal.org/licensing
7. Навчальні програми для 5-9 класів [Електронний ресурс] – Режим доступу до ресурсу: https://mon.gov.ua/ua/osvita/zagalna-serednya-osvita/navchalni-programi/navchalni-programi-5-9-klas
8. Інформатика 7 клас / Морзе, Н. В., Барна, О. В., Вембер, В. П., Кузмінська, О. Г. — К. Освіта, 2015. — c. 224.
9. Інформатика 7 клас / Ривкінд, Й. Я., Лисенко, Т. І., Чернікова, Л. А., Шакотько, В. В. — К. Генеза, 2015. — c. 256.
10. Scratch - Imagine, Program, Share [Електронний ресурс] – Режим доступу до ресурсу: https://scratch.mit.edu
11. Virtual Machine used to represent, run, and maintain the state of programs for Scratch 3.0 | GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/LLK/scratch-vm
12. ScratchEd [Електронний ресурс] – Режим доступу до ресурсу: http://scratched.gse.harvard.edu
13. Scratch Grades 6-8 [Електронний ресурс] – Режим доступу до ресурсу: https://www.finchrobot.com/teaching/scratch6-8
14. Інформатика 8 клас / Казанцева, О. П., Стеценко, І. В. — Т.Богдан, 2016. — c. 304.
15. Microsoft. Visual Basic 6.0 [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-basic-6/visual-basic-6.0-documentation
16. Visual Basic — Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Visual\_Basic
17. TIOBE Index [Електронний ресурс] – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/
18. Інформатика 8 клас / Гуржій, А. М., Карташова, Л. А., Лапінський, В. В. — Л.Світ, 2016. — c. 354.
19. C++ The Complete Reference Third Edition / Schildt, Herbert. — Osborne McGraw-Hill, 1998. — c. 1008.
20. The C++ Programming Language / Stroustrup, Bjarne. — Addison-Wesley, 1997. — c. 910.
21. Інформатика 8 клас / Морзе, Н. В., Барна, О. В., Вембер, В. П. — К.Оріон, 2016. — c. 240.
22. Python on Android [Електронний ресурс] – Режим доступу до ресурсу: https://www.damonkohler.com/2008/12/python-on-android.html
23. Інформатика 8 клас / Ривкінд, Й. Я., Лисенко, Т. І., Чернікова, Л. А., Шакотько, В. В. — К.Генеза, 2016. — c. 285.
24. Інформатика 8 клас / Бондаренко, О. О., Ластовецький, В. В., Пилипчук, О. П. — Х.Ранок, 2016. — c. 354.
25. lazarus-qt5 [Електронний ресурс] – Режим доступу до ресурсу: https://www.archlinux.org/packages/community/x86\_64/lazarus-qt5/
26. Lazarus project downloads on SourceForge (Lazarus Mac OS X i386 / Lazarus 1.8.2) [Електронний ресурс] – Режим доступу до ресурсу: https://sourceforge.net/projects/lazarus/files/Lazarus%20Mac%20OS%20X%20i386/Lazarus%201.8.2/
27. Lazarus project downloads on SourceForge (Lazarus Windows 64 bits / Lazarus 1.8.2) [Електронний ресурс] – Режим доступу до ресурсу: https://sourceforge.net/projects/lazarus/files/Lazarus%20Windows%2064%20bits/Lazarus%201.8.2/
28. Lazarus, Documentation. TThread [Електронний ресурс] – Режим доступу до ресурсу: https://www.freepascal.org/docs-html/rtl/classes/tthread.html
29. Lazarus, Wiki. Lazarus Documentation [Електронний ресурс] – Режим доступу до ресурсу: http://wiki.lazarus.freepascal.org/Lazarus\_Documentation

# **ДОДАТКИ**

Додаток А



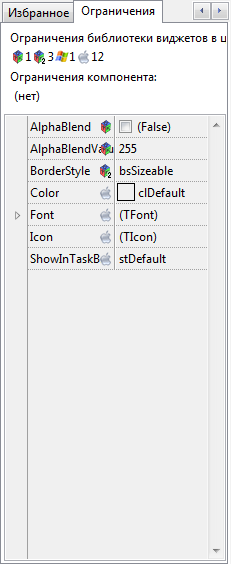
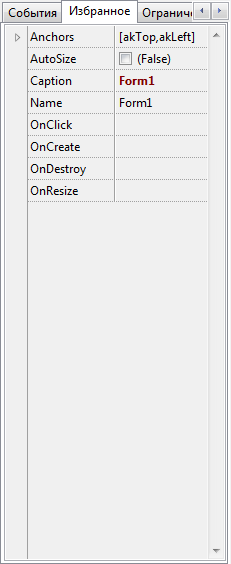
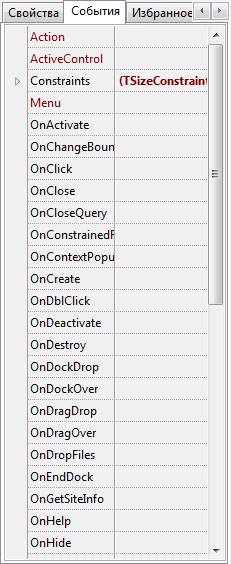
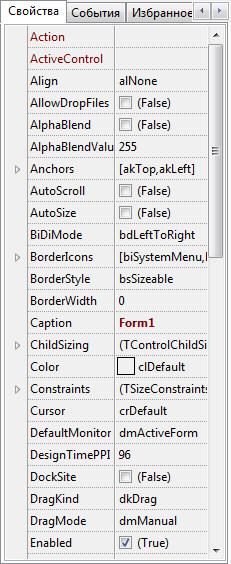
**Додаток Б**



**Додаток В**

D:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-17-27.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-18-13.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-18-55.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-19-04.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-19-18.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-19-26.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-19-36.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-19-44.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-19-52.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-20-00.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-20-08.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-20-15.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-20-22.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-20-29.pngD:\Диплом\Обрезать\верхнее меню\cropped\2018-04-21_09-20-36.png

**Додаток Г**



**Додаток Д**

