

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра математики та інформатики

Пістек Данило Ігоревич

**ВИКОРИСТАННЯ РЕДАКТОРА БЛОК-СХЕМ АЛГОРИТМІВ У
НАВЧАННІ ОСНОВ АЛГОРИТМІЗАЦІЇ В ЗАКЛАДАХ ЗАГАЛЬНОЇ
СЕРЕДНЬОЇ ОСВІТИ**

кваліфікаційна робота

здобувача вищої освіти другого (магістерського) рівня

освітньої програми «Інформатика»

за спеціальністю 014.09 .Середня освіта (Інформатика)

Особистий підпис _____  _____ Данило ПІСТЕК

Науковий керівник _____ Юрій КОЗУБ,
доктор технічних наук, професор
кафедри математики та інформатики

В.о. завідувача кафедри _____ Юрій КОЗУБ,
доктор технічних наук, професор
кафедри математики та інформатики

АНОТАЦІЯ

Пістек Д. І.

Тема: Використання редактора блок-схем алгоритмів у навчанні основ алгоритмізації в закладах загальної середньої освіти.

Спеціальність: 014.09 «Середня освіта (Інформатика)».

Установа: ЛНУ імені Тараса Шевченка, 2026р.

Магістерська робота містить: 103 с., 30 рис. 2 таб., 25 джерел.

Об'єкт дослідження - процес навчання основ алгоритмізації та програмування учнів у закладах загальної середньої освіти.

Предмет дослідження - програмні засоби візуалізації алгоритмів та методика їх використання при розв'язуванні задач з інформатики.

Мета дослідження - підвищення якості засвоєння фундаментальних алгоритмічних структур учнями шляхом створення та впровадження спеціалізованого візуального редактора блок-схем алгоритмів із відповідним методичним супроводом практичної підготовки.

Результати роботи. У роботі розглянуто методику викладання основ алгоритмізації в шкільному курсі інформатики. Проведено аналітичний огляд цифрових інструментів, що використовуються для побудови блок-схем.

Проведено аналіз процесу розробки програмного забезпечення редактора блок-схем алгоритмів. Проведено моделювання та описана архітектура розробленого додатка. Розроблено редактор блок-схем алгоритмів.

Розглянуто методичні підходи до використання редактора блок-схем алгоритмів у курсі інформатики. Наведено розробки п'яти практичних робіт з використанням розробленого редактора блок-схем алгоритмів.

Ключові слова: АЛГОРИТМІЧНЕ МИСЛЕННЯ, ОСНОВИ АЛГОРИТМІЗАЦІЇ, ПРОГРАМУВАННЯ, БЛОК-СХЕМА, ВІЗУАЛЬНЕ МОДЕЛЮВАННЯ, АЛГОРИТМ, QT4, PASCAL, C / C ++, МОВА ПРОГРАМУВАННЯ

ANNOTATION

Pistek Danylo

Theme Use of Using an algorithm flowchart editor in teaching the basics of algorithmization in secondary education institutions.

Speciality: 014.09 "Secondary Education (Informatics)".

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2026 year.

Master's work of: 103 p., 30 im, 25 sources.

Object of research – the process of teaching students the basics of algorithmization and programming in secondary education institutions.

Subject of research - software tools for visualization of algorithms and methods for their use in solving computer science problems.

Purpose of work – improving the quality of students' mastery of fundamental algorithmic structures by creating and implementing a specialized visual editor of algorithm flowcharts with appropriate methodological support for practical training.

Results of work. The paper considers the methodology for teaching the basics of algorithmization in a school computer science course. An analytical review of digital tools used to build flowcharts is conducted.

The process of developing software for an algorithm flowchart editor is analyzed. The architecture of the developed application is modeled and described. An algorithm flowchart editor is developed.

Methodological approaches to using an algorithm flowchart editor in a computer science course are considered. The development of five practical works using the developed algorithm flowchart editor is presented.

Keywords: ALGORITHMIC THINKING, BASICS OF ALGORITHMISATION, PROGRAMMING, BLOCK DIAGRAM, VISUAL MODELING, ALGORITHM, QT4, PASCAL, C / C ++, PROGRAMMING LANGUAGE

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДИЧНІ ЗАСАДИ НАВЧАННЯ	
АЛГОРИТМІЗАЦІЇ В ЗАКЛАДАХ ЗАГАЛЬНОЇ СЕРЕДНЬОЇ ОСВІТИ ..	10
1.1. Методика викладання основ алгоритмізації в шкільному курсі інформатики.....	10
1.2. Неперервність змістової лінії основ алгоритмізації та програмування	17
1.3. Вибір мови програмування для розділу «Основи алгоритмізації та програмування»	18
1.4. Аналіз підручників з теми «Основи алгоритмізації та програмування»	20
1.5. Аналітичний огляд цифрових інструментів, що використовуються для побудови блок-схем	22
Висновки до розділу	33
РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА РЕАЛІЗАЦІЯ РЕДАКТОРА БЛОК-СХЕМ АЛГОРИТМІВ	34
2.1. Логічне представлення моделі поведінки програмної розробки	34
2.2. Логічне уявлення статичної моделі структури програмної розробки	35
2.3. Математичний опис програми.....	39
2.3.1. Опис моделі даних	39
2.3.2. Математичний опис використовуваних моделей даних	43
2.3.3. Опис структур даних	44
2.4. Обґрунтування вибору середовища розробки системи для вирішення завдань	45
2.5. Алгоритм програми.....	49
2.6. Опис роботи редактора блок-схем алгоритмів	52
Висновки до розділу	56

РОЗДІЛ 3. РОЗРОБКА ПРАКТИЧНИХ РОБІТ ЗА ТЕМОЮ «АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ» З ВИКОРИСТАННЯМ РЕДАКТОРА БЛОК-СХЕМ АЛГОРИТМІВ.....	57
3.1. Методичні підходи до використання редактора блок-схем алгоритмів у курсі інформатики.....	57
3.2. Розробка практичної роботи за темою «Інструменти побудови алгоритму та принципи розробки блок-схем»	59
3.3. Розробка практичної роботи за темою «Базова структура алгоритму – лінійна».....	65
3.4. Розробка практичної роботи за темою «Базова структура алгоритму – розгалуження»	71
3.5. Розробка практичної роботи за темою «Базова структура алгоритму – циклічна».....	76
3.6. Розробка практичної роботи за темою «Основні алгоритми для обчислення суми та підрахунку кількості елементів масивів даних»	86
Висновки до розділу	97
ВИСНОВКИ	98
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	101
ДОДАТОК А.....	104

ВСТУП

У сучасній системі підготовки здобувачів освіти на уроках інформатики провідну роль відіграє розвиток алгоритмічного мислення, яке є базою для подальшого опанування програмування. Оскільки алгоритми лежать в основі будь-якої програмної діяльності, здатність аналізувати задачі, мислити алгоритмічно та наочно відображати логіку їх розв'язання є важливою складовою професійної компетентності здобувачів освіти.

Водночас у практиці загальної середньої освіти нерідко спостерігається несистемний підхід до формування алгоритмічного мислення, що часто зводиться до формального розв'язування задач або заучування конструкцій програмного коду без глибокого усвідомлення логіки їх побудови.

Одним із дієвих шляхів подолання цієї проблеми є застосування засобів візуального моделювання, зокрема використання блок-схем для графічного подання алгоритмів. Блок-схеми виступають інструментом структурованого мислення, який дає змогу представити складну послідовність дій у вигляді логічно пов'язаних елементів і переходів між ними. Такий підхід не лише полегшує розуміння алгоритмічних конструкцій, а й сприяє розвитку навичок декомпозиції задач, аналізу умов, циклів і розгалужень, що є необхідними для ефективного навчання програмуванню в шкільному курсі інформатики.

Проблема формування алгоритмічного мислення здобувачів освіти на уроках інформатики набуває особливої актуальності в умовах активного розвитку цифрових технологій та зростання ролі програмування в загальній середній освіті. Дослідники зазначають, що результативне навчання програмуванню має починатися не з вивчення синтаксису мов програмування, а з формування вміння логічно осмислювати задачу та представляти її у вигляді чітко структурованого алгоритму. У цьому контексті блок-схеми розглядаються як ефективний засіб візуального моделювання, що забезпечує доступність навчального матеріалу та стимулює розвиток мислення учнів і студентів [1; 8].

Блок-схеми дають можливість відобразити логіку виконання алгоритму через систему умов, циклів, послідовних і паралельних дій. Їх використання як

проміжного етапу між формулюванням задачі та її програмною реалізацією допомагає здобувачам освіти глибше усвідомити структуру майбутньої програми, сприяє формуванню навичок декомпозиції та вибору оптимальних керуючих структур.

У працях [5;7] підкреслюється, що опанування основ алгоритмізації потребує спеціальної методичної підтримки, яка забезпечує наочність і практичну спрямованість навчання. Автори наголошують на важливості формалізації задач у вигляді структурованих схем та необхідності відповідного дидактичного інструментарію в педагогічній підготовці. Науковці також звертають увагу на взаємозв'язок математичної й алгоритмічної підготовки, зазначаючи, що графічні представлення алгоритмів сприяють встановленню міждисциплінарних зв'язків і формуванню операційного стилю мислення [3]. Дослідження показують, що учні, які працювали з візуальними моделями алгоритмів, демонструють кращі результати під час переходу до текстового програмування [4]. Крім того, візуальна складова розглядається як ефективний засіб підвищення мотивації до вивчення програмування [6], оскільки робота з графічними елементами, зокрема з блок-схемами, допомагає краще усвідомити взаємозв'язок між діями та результатами виконання програм. Важливо, що здобувачі освіти, які мали досвід використання таких інструментів, виявляють вищий рівень методичної готовності до програмування.

Разом з тим, упровадження блок-схем у навчальний процес потребує наявності відповідного програмного забезпечення для візуального моделювання алгоритмів.

Аналіз існуючих практик засвідчує брак уваги до системного й цілеспрямованого використання спеціалізованих середовищ для побудови блок-схем у курсах фахової підготовки на уроках інформатики.

Це свідчить про необхідність розробки методичних підходів, які забезпечуватимуть ефективне використання таких засобів у процесі розв'язування типових обчислювальних задач.

Об'єкт дослідження - процес навчання основ алгоритмізації та програмування учнів у закладах загальної середньої освіти.

Предмет дослідження - програмні засоби візуалізації алгоритмів та методика їх використання при розв'язуванні задач з інформатики.

Мета дослідження - підвищення якості засвоєння фундаментальних алгоритмічних структур учнями шляхом створення та впровадження спеціалізованого візуального редактора блок-схем алгоритмів із відповідним методичним супроводом практичної підготовки.

Досягнення зазначеної мети передбачає вирішення таких основних завдань:

- Розглянути методику викладання основ алгоритмізації в шкільному курсі інформатики;
- Провести аналітичний огляд цифрових інструментів, що використовуються для побудови блок-схем;
- Провести моделювання та аналіз програмного забезпечення для середовища розв'язання задач з програмування, розробити математичну модель для редактора блок-схем алгоритмів;
- Розробити та реалізувати систему візуального проектування редактора блок-схем алгоритмів;
- Розробити практичні роботи за темою «алгоритмізація та програмування» з використанням редактора блок-схем алгоритмів.

Методологічну базу дослідження утворюють принципи когнітивної візуалізації, ідеї візуально орієнтованого навчання та положення педагогіки алгоритмізації. Теоретичне підґрунтя роботи сформовано на основі аналізу наукових джерел, у яких розглядаються питання розвитку алгоритмічного мислення, методики навчання програмування, а також застосування візуалізації алгоритмів у закладах загальної середньої освіти. У процесі дослідження використано методи теоретичного аналізу фахової літератури з метою визначення сучасного стану проблеми та обґрунтування доцільності й результативності візуального підходу в навчанні алгоритмізації.

У першому розділі розглянуто методику викладання основ алгоритмізації в шкільному курсі інформатики. Проведено аналітичний огляд цифрових інструментів, що використовуються для побудови блок-схем.

У другому розділі проведено аналіз процесу розробки програмного забезпечення редактора блок-схем алгоритмів. Проведено моделювання та описана архітектура розробленого додатка. Описано математичну модель редактора блок-схем алгоритмів. Розроблено редактор блок-схем алгоритмів. Розроблений прототип інтерактивної побудови блок-схем дозволяє шляхом маніпулювання графічними об'єктами редагувати логіку програм, автоматично формувати програмні коди для різних мов програмування (Pascal, C / C ++, Алгоритмічна мова). Програма написана на мові C ++ на основі бібліотеки Qt 4.

У третьому розділі розглянуто методичні підходи до використання редактора блок-схем алгоритмів у курсі інформатики. Наведено розробки п'яти практичних робіт за темами: «Інструменти побудови алгоритму та принципи розробки блок-схем», «Базова структура алгоритму – лінійна», «Базова структура алгоритму – розгалуження», «Базова структура алгоритму – циклічна», «Основні алгоритми для обчислення суми та підрахунку кількості елементів масивів даних» з використанням розробленого у другому розділі редактора блок-схем алгоритмів.

РОЗДІЛ 1.

ТЕОРЕТИКО-МЕТОДИЧНІ ЗАСАДИ НАВЧАННЯ АЛГОРИТМІЗАЦІЇ В ЗАКЛАДАХ ЗАГАЛЬНОЇ СЕРЕДНЬОЇ ОСВІТИ

1.1. Методика викладання основ алгоритмізації в шкільному курсі інформатики

Сучасна система освіти спрямована на виховання освічених та інтелектуально розвинених особистостей, які мають цілісне бачення світу й глибоке усвідомлення взаємозв'язків між явищами. Випускники закладів загальної середньої освіти повинні бути здатними ефективно взаємодіяти з навколишнім середовищем, оперативно адаптуватися до змін і демонструвати компетентність у провідних сферах життєдіяльності, зокрема праці, державного управління, сімейних відносин, збереження здоров'я, політики та культури. Рівень функціональної грамотності в зазначених сферах є показником загальної компетентності особистості [8].

На сьогодні інформатика посідає провідне місце серед навчальних предметів, оскільки сприяє розвитку логічного мислення та формуванню вмінь ефективної роботи з інформацією. Вона забезпечує можливість орієнтуватися в умовах стрімкого розвитку цифрових технологій. Водночас динамічність цих процесів ускладнює своєчасне засвоєння нововведень як учнями, так і вчителями, що нерідко зумовлює поверхове опанування окремих аспектів навчального матеріалу.

Основною метою вивчення навчальної дисципліни «Інформатика» є формування в учнів практичних навичок у сфері інформаційних технологій, необхідних для створення комп'ютерних інформаційних продуктів. Реалізація цієї мети передбачає впровадження в освітній процес спеціально розробленої системи завдань, зорієнтованих на розв'язання наближених до реальних професійних проблем, з якими стикаються фахівці різних галузей. Це вимагає застосування адекватних педагогічних підходів і методів навчання. Формування інформаційної компетентності старшокласників ґрунтується на принципах науковості, системності та послідовності, поєднання навчання з практичною

діяльністю, розвитку самосвідомості учнів, диференціації та індивідуалізації навчання, доступності, міцності засвоєння знань, навчання на високому рівні складності та наочності [12].

Питання навчання інформатики в закладах загальної середньої освіти України висвітлено в наукових працях низки дослідників, зокрема Гевка І., Копосова Д., Донченка Я., Бурдуна О., Камалова Р., Гурняка І., Березовської Ю., Абушкіна Х..

Вітчизняні науковці Андрєєв Д., Бондаренко В., Ащепкова Н., Бугайчук К., Голощанов А. наголошують, що стрімкий розвиток технологій і вдосконалення технічних засобів не супроводжуються їх масовим оновленням у закладах освіти України в найближчій перспективі, що потребує адаптації змісту та методів навчання до наявних умов.

Особливу роль у шкільному курсі інформатики відіграє розділ «Основи алгоритмізації та програмування». Він розкриває значення алгоритмів та їхню ключову роль у взаємозв'язку понять «інформація», «алгоритм» і «комп'ютер», що лежить в основі процесу автоматизованого опрацювання даних. Розгляд прикладів забезпечує можливість формального виконання алгоритмів, наочно демонструє послідовність дій, які має здійснювати виконавець відповідно до заданих інструкцій. Це підкреслює здатність передавати виконання формалізовано описаних алгоритмів машині-виконавцю, тобто здійснювати автоматизацію діяльності людини на алгоритмічній основі. Опанування алгоритмічної мови сприяє ознайомленню учнів із формальними способами подання алгоритмів і розширює їхні уявлення про засоби їх опису. Метою навчання алгоритмізації є формування в учнів знань і практичних умінь щодо основних способів організації операцій і даних, а також опанування базових алгоритмічних конструкцій для побудови алгоритмів розв'язання різноманітних навчальних і практичних завдань.

В умовах реалізації Концепції Нової української школи особливої актуальності набуває формування в здобувачів освіти ключових і предметних компетентностей, зокрема інформаційно-цифрової та інноваційної. Оволодіння

основами алгоритмізації та програмування сприяє розвитку логічного, критичного й алгоритмічного мислення, уміння аналізувати проблеми, планувати діяльність і знаходити ефективні шляхи їх розв’язання, що відповідає компетентнісній парадигмі сучасної освіти.

Здобуття знань і практичних навичок у сфері алгоритмізації та програмування створює умови для самореалізації учнів, розвитку їх творчого потенціалу та формування готовності до використання цифрових технологій у навчальній і повсякденній діяльності. У контексті НУШ алгоритмізація та програмування розглядаються не лише як складова предметного змісту інформатики, а як ефективний засіб розвитку інноваційного потенціалу особистості учня та професійної компетентності вчителя.

Відповідно до Державного стандарту базової середньої освіти, вивчення алгоритмізації та програмування є наскрізною змістовою лінією курсу інформатики та реалізується поетапно на різних рівнях навчання. Такий підхід забезпечує поступове й системне формування алгоритмічного мислення, починаючи з початкової школи та продовжуючи в основній і старшій школі.

У межах шкільного курсу інформатики алгоритмізація реалізується за двома взаємопов’язаними напрямками:

- **розвивальним**, спрямованим на формування алгоритмічного, логічного та критичного мислення учнів;
- **практико-орієнтованим**, що передбачає опанування основ програмування та навичок створення простих програмних продуктів.

Практико-орієнтований напрям охоплює фундаментальний аспект, який формує уявлення про мови програмування, структуру програм і принципи їх розроблення, а також профорієнтаційний аспект, що дозволяє учням усвідомити власні здібності й інтереси до ІТ-сфери та сприяє свідомому вибору подальшої освітньої траєкторії.

Алгоритмізація у шкільному курсі інформатики відповідає принципам структурного програмування та є підготовчим етапом до вивчення об’єктно-

орієнтованого програмування, що узгоджується з вимогами сучасного цифрового суспільства та ринку праці.

Зміст навчання з алгоритмізації передбачає послідовне опрацювання базових алгоритмічних структур: лінійних алгоритмів, алгоритмів із розгалуженнями, циклічних алгоритмів, а також використання допоміжних алгоритмів і процедур. У межах НУШ важливе значення надається діяльнісному підходу, що реалізується через розв'язування практичних задач і проєктну діяльність.

У процесі навчання основ алгоритмізації доцільно акцентувати увагу на таких ключових аспектах:

1. Виявлення загальних закономірностей і принципів алгоритмізації.

Передбачає ознайомлення з основними етапами розв'язування задач із використанням сучасних інформаційних технологій, аналіз умов задачі, способів її формалізації та моделювання реальних процесів. Важливим є також вибір виконавця з урахуванням його характеристик і допустимих операцій, а також застосування методів і засобів формалізованого опису дій виконавця за допомогою комп'ютера. Однією з актуальних проблем навчання цього розділу є поєднання традиційної алгоритмічної лінії курсу з більш динамічними та інноваційними підходами до виконавців, формалізації, моделювання та використання інформаційних технологій.

2. Алгоритмізація як складова теоретичної інформатики.

Оскільки створення алгоритмів має фундаментальний характер, алгоритмізація належить до теоретичної інформатики. Водночас розвиток інформаційних технологій, зокрема програмування, створює умови для ознайомлення учнів у межах розділу «Основи алгоритмізації» з сучасними та перспективними напрямками цієї галузі [13].

3. Теорія та методика навчання алгоритмізації.

Методика викладання алгоритмізації в шкільному курсі інформатики повинна охоплювати процес навчання на всіх рівнях і в різних формах: у

загальноосвітній школі, на різних щаблях середньої освіти, у процесі самостійного опанування інформатики, а також у дистанційному навчанні. Кожен із цих напрямів формує специфічні завдання для сучасної педагогічної науки. Водночас особливе значення має методологія інформатики, що досліджує навчання інформатики в закладах загальної середньої освіти як складову загальної інформатичної підготовки [8].

Сучасне розуміння навчання основ алгоритмізації ґрунтується на низці провідних принципів [8]:

1. Спрямованість на використання комп'ютера.

Вивчення основ алгоритмізації орієнтоване на застосування комп'ютера як дієвого та універсального засобу навчальної діяльності.

2. Зосередження на змістовній сутності алгоритму.

Основна мета навчання полягає у виокремленні змісту алгоритму та усвідомленні правил його побудови. При цьому акцент робиться не на засвоєнні конкретних алгоритмічних мов, а на розумінні їх як одного з можливих інструментів формального подання алгоритмів.

3. Формування операційного мислення.

Поняття алгоритму є базовим для розвитку алгоритмічного мислення та усвідомлення учнями можливостей автоматизації різних видів діяльності. Це сприяє розширенню уявлень про формальний і механічний характер дій під час виконання алгоритмів, що лежить в основі автоматизації операцій за допомогою комп'ютерних засобів.

4. Інтеграція теоретичних знань і практичної діяльності.

Оволодіння поняттям «алгоритм» виступає початковим етапом формування уявлень про автоматизоване опрацювання даних за допомогою комп'ютера. Водночас важливим є врахування внутрішньопредметних зв'язків і підкреслення значущості формалізованого опису алгоритмів [4].

5. Ознайомлення з теорією алгоритмів.

Поняття алгоритму належить до фундаментальних математичних категорій і є предметом дослідження теорії алгоритмів, що зумовлює його важливе місце в змісті навчання інформатики.

До основних компонентів навчання належать:

- опанування відомих алгоритмів і практики їх застосування;
- вивчення класичних алгоритмів;
- формування вмінь конструювати та описувати алгоритми як із використанням готових алгоритмічних рішень, так і без їх залучення.

Важливу роль у процесі розв'язання навчальних завдань відіграє алгоритмічна мова, яка забезпечує можливість подання алгоритмів за уніфікованими правилами. Опанування такої мови є необхідним етапом у навчанні основ алгоритмізації, оскільки сприяє формуванню навичок формалізованого опису алгоритмів.

На сучасному етапі розвитку теорії та методики навчання основ алгоритмізації у шкільному курсі інформатики спостерігається їх інтенсивне становлення та вдосконалення. Незважаючи на те, що шкільна інформатика функціонує вже майже два десятиліття, низка завдань, пов'язаних із новітніми напрямками педагогічної науки, сформувалася відносно недавно й потребує подальшого ґрунтовного теоретичного осмислення та тривалих експериментальних досліджень.

У межах загальних цілей викладання інформатики методика навчання основ алгоритмізації в середній школі передбачає розв'язання таких основних завдань [15]: визначення конкретних цілей навчання та його місця в навчальному плані закладів загальної середньої освіти; розроблення й обґрунтування ефективних методів і форм навчальної діяльності для досягнення визначених цілей; аналіз і добір усього спектра дидактичних засобів навчання основ алгоритмізації та підготовка рекомендацій щодо їх практичного використання в діяльності вчителя.

У низці наукових праць наголошується, що впродовж тривалого часу підготовка майбутніх учителів інформатики характеризується недостатнім

рівнем уваги, насамперед у сфері методичної підготовки. Визначення змісту навчальної дисципліни передбачає охоплення загальнотеоретичних засад викладання інформатики, поєднання програмно-технічних засобів, а також розроблення конкретної методики опрацювання окремих тем курсу інформатики на всіх етапах освітнього процесу.

Методика навчання основ алгоритмізації є порівняно молодого науковою галуззю, однак вона сформувалася не ізольовано. Як самостійна дисципліна, вона інтегрує знання з різних наук, зокрема інформатики, педагогіки, фізіології, філософії, психології, вікової педагогіки, а також узагальнений практичний досвід викладання інших навчальних предметів. Сучасне розуміння викладання основ алгоритмізації спирається на міждисциплінарні відомості з біології (дослідження саморегульованих біологічних систем, зокрема людини та інших живих організмів), історії й соціальних наук (аналіз соціальних систем), української мови (граматика, синтаксис, семантика), логіки (мисленнєві процеси, формальні операції, поняття істини та хиб), математики (числа, змінні, функції, відношення, операції) та психології (мислення, комунікація) [11].

Умови глобальної цифровізації різних сфер людської діяльності та зростаючого впливу інформатики на інші галузі знань дають підстави стверджувати, що методика навчання основ алгоритмізації має тісні міждисциплінарні зв'язки практично з усіма науками. Це особливо актуально в контексті переходу системи загальної середньої освіти України до профільного навчання, за якого факультативні курси з основ алгоритмізації набувають значущості для всіх освітніх профілів і навчальних предметів шкільної програми [12]. У такій ситуації об'єктом навчання стають не лише поняття та методи інформатики, а й елементи інших наук, що інтегруються з інформатикою в межах факультативних курсів.

Фахівець у галузі комп'ютерних наук повинен орієнтуватися в широкому спектрі дисциплін, зокрема у філософії (для формування цілісного світоглядного підходу до системно-інформаційної картини світу), філології та лінгвістиці (для розуміння принципів функціонування мов програмування, текстових редакторів,

систем розпізнавання тексту, машинного перекладу та штучного інтелекту), математиці, фізиці й економіці (для задач комп'ютерного моделювання), мистецтві (для роботи з графічними редакторами, дизайном і мультимедійними системами) тощо. Відповідно, учитель інформатики має володіти широким колом знань і постійно підвищувати рівень своєї професійної підготовки та кваліфікації [15].

Результати досліджень свідчать, що на початкових етапах вивчення алгоритмічних структур доцільним є застосування візуальних засобів навчання, зокрема блок-схем, які забезпечують наочне подання логіки алгоритмів та полегшують їх розуміння. Використання спеціалізованих програмних середовищ для побудови та моделювання блок-схем відповідає принципам НУШ, оскільки сприяє активній пізнавальній діяльності учнів, формуванню цифрової грамотності та розвитку здатності до самостійного конструювання алгоритмів.

1.2. Неперервність змістової лінії основ алгоритмізації та програмування

Інформатика як навчальний предмет у закладах загальної середньої освіти розпочинає вивчатися з другого класу, при цьому однією з її ключових змістових ліній є основи алгоритмізації та програмування. Доцільно окремо проаналізувати особливості навчання в 5–9-х і 10–11-х класах.

У 5–9-х класах основи алгоритмізації та програмування подаються у межах окремого розділу «Алгоритми та програми», який є лише складовою шкільного курсу інформатики. Така фрагментарність призводить до перерв у вивченні відповідного матеріалу, унаслідок чого учні поступово втрачають сформовані раніше знання та вміння (наприклад, у 8-му класі не зберігаються навички, набуті в 7-му). Ця проблема має системний характер і потребує комплексного розв'язання. Одним із можливих шляхів її подолання є виокремлення основ алгоритмізації та програмування в самостійну навчальну дисципліну, що, своєю чергою, порушує низку дискусійних питань щодо обсягу навчального часу, віку початку навчання та змістового наповнення курсу.

Водночас на сучасному етапі система шкільної інформатичної освіти не готова до впровадження таких радикальних змін у найближчій перспективі.

У 10–11-х класах інформатика викладається на рівні стандарту та на профільному рівні. Слід зазначити, що на рівні стандарту в інваріантній складовій навчального плану не передбачено вивчення основ алгоритмізації та програмування; опанування цих питань можливе лише в межах вибіркового модуля «Креативне програмування».

На профільному рівні зміст навчання інформатики структуровано за двома основними змістовими лініями:

1. основи алгоритмізації та програмування (зокрема об'єктно-орієнтований підхід);
2. інформаційні технології.

Відповідно до програми профільного рівня [7], на вивчення інформатики відводиться 5 годин на тиждень. Питання алгоритмізації та програмування розглядаються у розділах «Мова програмування та структури даних» (10 клас), «Алгоритми» та «Парадигми та технології програмування» (11 клас), при цьому основний акцент робиться на опануванні мов програмування й сучасних технологій програмування.

Згідно з чинною програмою з інформатики профільного рівня, учитель має право самостійно визначати кількість навчальних годин, відведених на вивчення окремих розділів і тем, а також послідовність їх опрацювання [7]. З огляду на це доцільно організувати паралельне вивчення змістових ліній, наприклад, 3 години на тиждень присвячувати основам алгоритмізації та програмування і 2 години — інформаційним технологіям. Такий підхід сприятиме забезпеченню неперервності формування алгоритмічного мислення та системності засвоєння відповідних знань і вмінь.

1.3. Вибір мови програмування для розділу Основи алгоритмізації та програмування»

Навчальні програми не регламентують конкретну мову програмування для вивчення, тому право вибору залишається за вчителем або закладом освіти. У

процесі навчання основ алгоритмізації та програмування застосовуються різні платформи й мови — від Pascal до Java чи C#. Загалом питання вибору мови програмування є складним і дискусійним. Ще у 2013 році редакція журналу «Комп'ютер у школі та сім'ї» ініціювала обговорення проблеми «Яку мову програмування вивчати у школі?». Результатом стали кілька публікацій [10; 11], у яких відомі педагоги представили аргументовані позиції щодо доцільності використання різних мов програмування, зокрема Pascal, VB, VBA, Python, C/C++, C#, Java та інших.

Ми вважаємо доцільним здійснювати вибір мови програмування з урахуванням вікових особливостей учнів, тобто застосовувати різні мови для 5–6-х, 7–9-х та 10–11-х класів. Окрім того, для обґрунтованого вибору мови програмування необхідно визначити відповідні критерії. На нашу думку, до них належать: актуальність мови; наявність безкоштовного транслятора; простота й зрозумілість синтаксису; підтримка різних парадигм програмування, зокрема структурного, функціонального та об'єктно-орієнтованого.

З урахуванням вікових особливостей учнів і визначених критеріїв пропонуємо у 5–6-х класах під час вивчення розділу «Алгоритми та програми» використовувати блочні візуальні середовища програмування, такі як Scratch, Google Blockly, MIT App Inventor або аналогічні. Доцільність такого вибору пояснюється тим, що у візуальних середовищах відсутня потреба у текстовому введенні програмного коду: створення програм здійснюється шляхом комбінування готових візуальних блоків. Це дає змогу реалізовувати основні алгоритмічні структури — лінійні, розгалужені та циклічні. Основною метою використання блочних середовищ є формування інтересу до алгоритмізації, а також забезпечення доступності й привабливості вивчення програмування для молодших школярів. Такий підхід, на нашу думку, створює ефективне підґрунтя для подальшого переходу до текстових мов програмування у 7–9-х класах.

У 7–9-х класах для опрацювання розділу «Алгоритми і програми» доцільно обрати мову Python. Такий вибір зумовлений низкою чинників. Python є сучасною та популярною мовою програмування і станом на травень 2024 року

посідає перше місце в рейтингу TIOBE Index. Мова має відкритий програмний код і поширюється за ліцензією GPL. Її синтаксис вирізняється простотою та зрозумілістю, а програмні реалізації, як правило, є коротшими порівняно з аналогами, створеними мовами Pascal чи C/C++. Python підтримує різні парадигми програмування, зокрема структурну, функціональну та об'єктно-орієнтовану. Для роботи з мовою розроблено декілька середовищ програмування, серед яких стандартне середовище IDLE, що входить до складу дистрибутива. Водночас певним обмеженням є відсутність у стандартному середовищі візуального конструктора для створення графічних інтерфейсів.

У 10–11-х класах на рівні стандарту в межах вивчення вибіркового модуля «Креативне програмування» також доцільно використовувати мову Python. За результатами досліджень [9], перспективними для вивчення у старшій школі вважаються такі популярні мови програмування, як C#, Java, Python та C++. На профільному рівні можливі різні підходи до навчання основ алгоритмізації та програмування. Один із них полягає у продовженні вивчення мови Python, проте з акцентом не на базові алгоритмічні конструкції, а на розроблення практично орієнтованих проєктів.

1.4. Аналіз підручників з теми «Основ алгоритмізації та програмування»

У підручниках з інформатики для 5–6 класів ознайомлення з елементами алгоритмізації та програмування, як правило, здійснюється з використанням мови Scratch. Водночас у шкільних підручниках з інформатики для 7–9 класів видання 2016 року мова Python майже не була представлена, за винятком навчальних посібників авторського колективу Н. В. Морзе. У більшості підручників того періоду як базову мову програмування було обрано Free Pascal із середовищем Lazarus. Разом із тим у підручниках (Морзе, Барна, Вембер, 2016), окрім зазначених мови та середовища, подано також відомості про Python із прикладами програмної реалізації.

Зокрема, у 8-му та 9-му класах під час вивчення розділів «Алгоритми роботи з об'єктами та величинами» та «Табличні величини та алгоритми їх опрацювання» відповідно передбачено використання мови Python.

Суттєві зміни відбулися у підручниках з інформатики для 7–9 класів, виданих у 2020 році. Автори навчальних видань відмовилися від використання мови Free Pascal та середовища Lazarus, зосередивши увагу переважно на вивченні мови Python. Так, у підручниках (Морзе, Барна, 2020) для 7 класу під час опрацювання розділу «Алгоритми і програми» поряд із Python подаються також елементи програмування у середовищі Scratch. Водночас у підручнику для 8 класу (Ривкінд, Лисенко, Чернікова, Шакотько, 2021) у межах розділу «Алгоритми і програми» пропонується використання як середовища Lazarus, так і мови Python.

Що стосується старшої школи, то, як уже зазначалося, на рівні стандарту вивчення програмування можливе лише в межах вибіркового модуля «Креативне програмування», для якого окремих підручників наразі не розроблено. У підручнику з інформатики для 10 класу профільного рівня (Руденко, Речич, Потієнко, 2019) у розділі «Мови програмування та структури даних» як основну мову програмування обрано Python. У ньому поряд з основами алгоритмізації та програмування подано також відомості з об'єктно-орієнтованого програмування, що реалізуються на прикладі мови Python та середовища програмування IDLE. Зокрема, розглядаються такі питання: структура та способи виконання проєктів мовою Python; оператори, вирази й засоби опрацювання числових даних; реалізація базових алгоритмічних конструкцій; вбудовані типи даних і методи їх опрацювання; користувацькі функції та модулі Python; основи об'єктно-орієнтованого програмування; елементи побудови графічного інтерфейсу користувача.

У підручнику для 11 класу профільного рівня (Руденко, Речич, Потієнко, 2019) у розділі «Алгоритми» автори пропонують вивчення та практичну реалізацію базових алгоритмів, зокрема алгоритмів сортування й пошуку даних, опрацювання рядків, роботи з графами, динамічного програмування, жадібних

алгоритмів і основ обчислювальної геометрії, із застосуванням мови Python та середовища IDLE.

Ще у 2013 році редакція журналу «Комп'ютер у школі та сім'ї» ініціювала фахову дискусію серед провідних учителів інформатики та науковців у галузі навчальної інформатики щодо питання вибору мови програмування для шкільного курсу. Результатом обговорення стала низка публікацій, у яких було представлено аргументовані позиції стосовно доцільності вивчення різних мов програмування, зокрема Pascal, C, C++, Visual Basic, Visual Basic for Applications, JavaScript, Python та інших. Узагальнюючи, можна зазначити, що кожна з наведених мов має свої переваги й обґрунтування щодо використання в освітньому процесі, однак остаточний вибір значною мірою залежить від професійної позиції вчителя. На думку авторів дослідження (Юрченко, Семеніхіна, Хворостіна, Удовиченко, Петренко, 2019), перспективним напрямом у старшій школі є вивчення сучасних і популярних мов програмування, зокрема Java, Python та C++.

1.5. Аналітичний огляд цифрових інструментів, що використовуються для побудови блок-схем

Для реалізації поставлених завдань необхідним є вивчення предметної області, а саме існуючих програм-редакторів блок-схем, їх переваг і недоліків.

Редактор блок-схем - спеціалізована програма, вона надає той набір інструментів, який необхідний саме для створення блок-схем, що є істотним аргументом на користь застосування даної програми, а не використання графічних редакторів. Набір додаткових опцій дозволяє оптимізувати процес розробки блок-схем і подальшого перетворення їх в процедури і функції мови програмування.

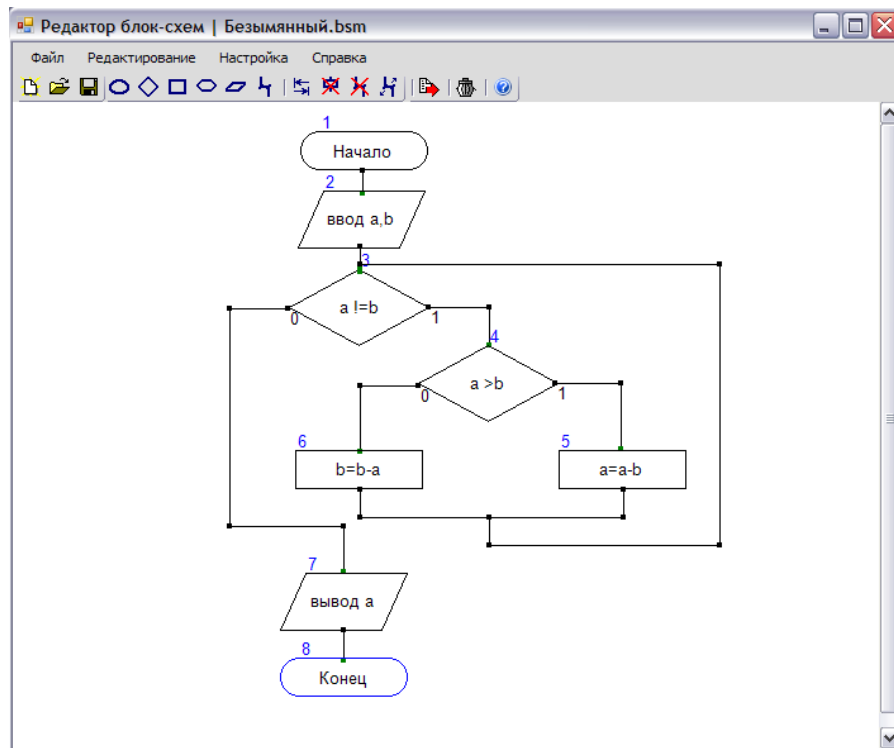


Рис. 1.1. Редактор блок-схем

Системні вимоги програми дуже скромні і вона запускається практично на будь-якому комп'ютері з будь-якою версією Windows.

Зупиняючись докладніше на опціях редактора, слід зазначити найбільш важливі з них:

1. Використання шаблонів при створенні блок-схем.
2. Імпорт процедур і функцій мов програмування. Редактор надає можливість імпортувати процедури і функції, реалізовані на якомусь з відомих мов програмування. Ця опція корисна для того, щоб краще розібратися в структурі алгоритму, написаного на мові програмування.
3. Експорт блок-схем в процедури і функції мов програмування.
4. Експорт блок-схем в різні графічні формати.

FCEditor. Заснована ідея цієї програми - зобразити блок-схему з блоків з довільним за величиною (мається на увазі текст) змістом. У більшості редакторів, якщо і є можливість автоматично змінювати розмір компонентів, то всі стрілки і переходи все одно треба розставляти вручну. У FCEditor це робиться автоматично.

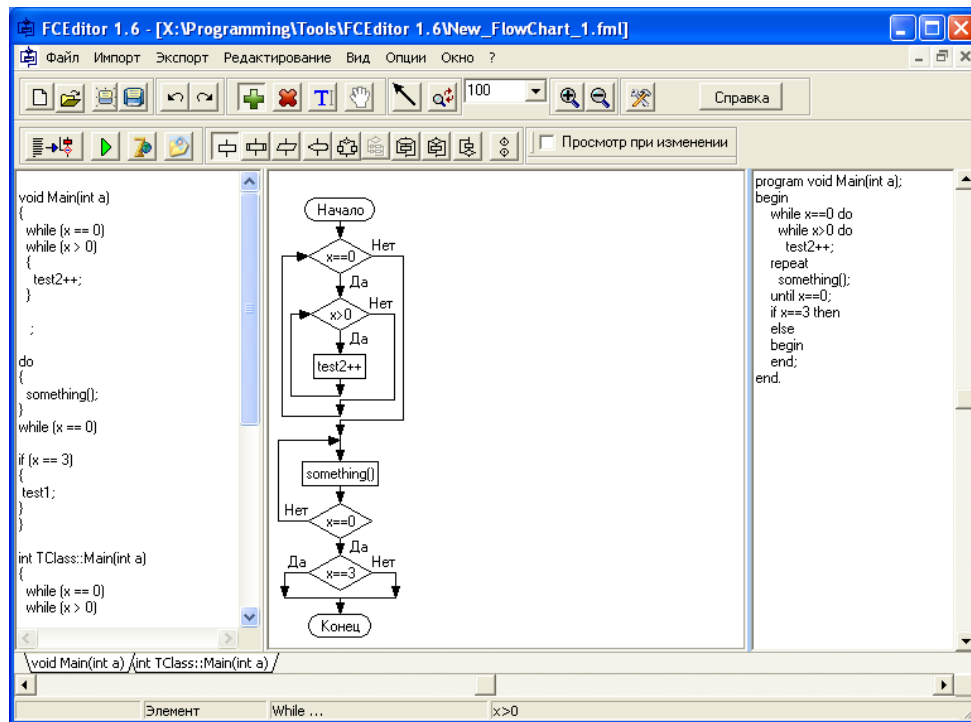


Рис. 1.2. FCEditor

Можливості FCEditor:

1. імпорт схеми з програмного коду;
2. автоматичне вирівнювання блоків і стрілок;
3. можливість зміни типу блоків;
4. копіювання і вставка блоків схеми;
5. окрема схема для кожної процедури;
6. можливість вставки розривів сторінок;
7. експорт схеми в графічний файл;
8. експорт схеми в код.

Таким чином, незважаючи на невелику вагу FCEditor є досить багатофункціональною програмою-редактором схем.

Розглянувши існуючі програми-редактори схем, ми можемо їх узагальнити, кожна з них:

1. Чи може використовувати шаблони для створення схем.
2. Може експортувати створену схему в різні формати.
3. Володіє автоматичної «підгонкою» блоків і стрілок.
4. Чи може створювати різні типи схем.

Visustin

Це автоматизований програмний засіб для побудови блок-схем, орієнтований на розробників програмного забезпечення та авторів технічної документації. Visustin здійснює зворотний інжиніринг вихідного коду, перетворюючи його на блок-схеми або UML-діаграми діяльності (Activity Diagram). Програма аналізує умовні оператори if/else, циклічні конструкції та оператори переходу й у повністю автоматичному режимі формує блок-схему, не потребуючи ручного втручання користувача.

Visustin підтримує створення блок-схем на основі програм, написаних багатьма мовами програмування, зокрема ABAP, ActionScript, Ada, ASP, різними асемблерними мовами, AutoIt, BASIC, пакетними файлами .bat, C, C++, C#, COBOL, Delphi, Fortran, HTML, Java, JavaScript, MATLAB, Pascal, Perl, PHP, Python, Ruby, SQL, VB, VBA, VB.NET та іншими.

Програма автоматично трансформує вихідний код у блок-схеми з оптимальною візуальною компоновкою, що забезпечує наочне та зрозуміле представлення алгоритму. Для створення схеми достатньо виконати одну дію, після чого система генерує готовий результат. Блок-схеми відображають структуру коду та, за потреби, коментарі до нього.

Visustin підтримує як класичні блок-схеми, так і UML-діаграми діяльності, а також надає можливість збереження результатів у форматі PDF, графічних файлів або у вигляді веб-сторінок [7].

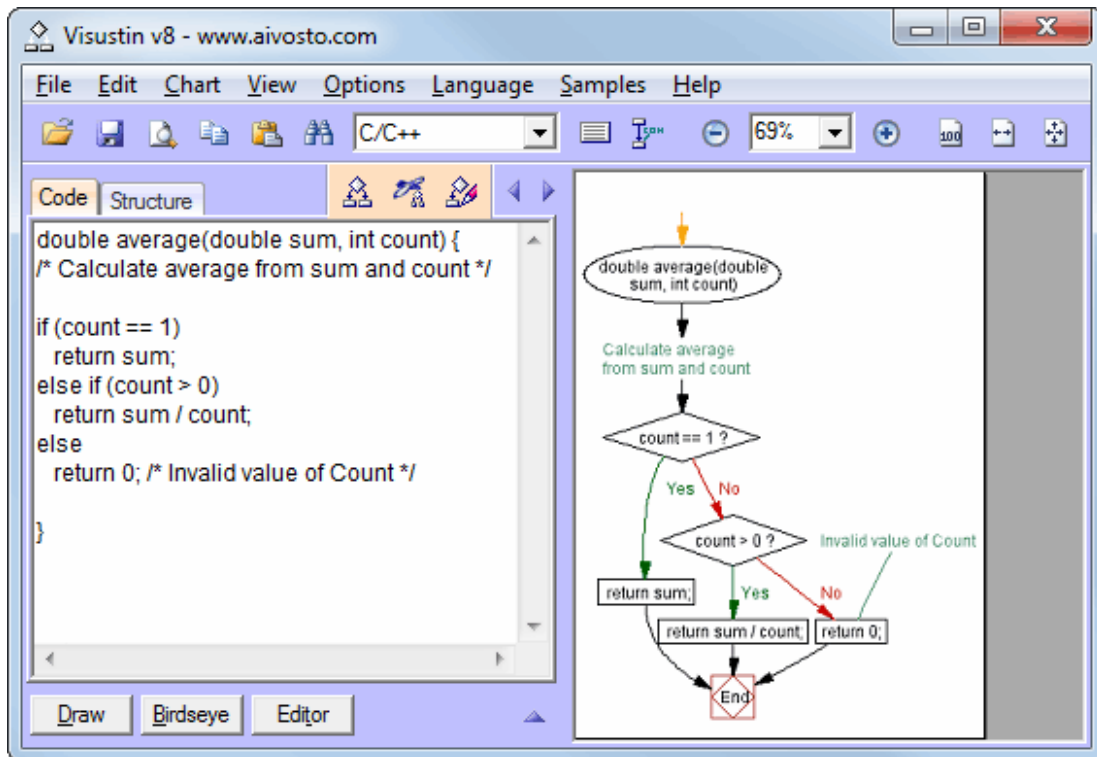


Рис. 1.3. Інтерфейс Visustin

Недоліки:

- інтерфейс програми реалізований англійською мовою;
- програмне забезпечення поширюється на комерційній основі (є платним).

Code2Flow

Однак у проаналізованих програм-редакторів схем є недоліки - вони не пристосовані для українських графічних стандартів. Жодна з розглянутих програм не має функцію автоматичного розміщення блоків і не дозволяє отримати код програми відповідної побудованої блок схемою.

Проектована нами програма повинна володіти позитивними властивостями, описаними вище, а також повинна мати можливість будувати схеми за українськими стандартами, наприклад ГОСТ 19.701. Так само в створювану програму буде включено інтерпретатор програмного коду, згенерованого програмою, з його допомогою програму можна буде використовувати для вирішення алгоритмічних задач або для навчання програмуванню. Алгоритм програми можна буде редагувати в візуальному режимі, а не в режимі набору програмного коду. Потім за допомогою програми

цей код можна буде виконати і перевірити результат роботи алгоритму. Це істотно підвищить ефективність процесу навчання програмуванню на початковому етапі.



Рис. 1.4. Інтерфейс Code2Flow

Переваги:

- створення блок-схем не потребує використання миші, що суттєво пришвидшує робочий процес;
- відсутня необхідність опановувати окрему мову програмування — застосовуються прості та зрозумілі правила синтаксису;
- наявна вбудована довідкова система з прикладами;
- підтримується експорт результатів у формати PDF, SVG та PNG.

Недоліки:

- інтерфейс реалізований виключно англійською мовою;
- у безкоштовній версії доступне створення не більше трьох блок-схем;
- обмежений функціонал: не підтримується опрацювання масивів функцій;
- функції, що не були задекларовані у початкових визначеннях класу або об'єкта (наприклад, додані пізніше), здебільшого не аналізуються;

- у мові Python не обробляються успадковані від батьківських класів функції та конструкції імпорту типу `import ... as ...`;
- у JavaScript використання прототипів може призводити до некоректних або непередбачуваних результатів.

Crystal Flow

CRYSTAL FLOW — це прикладне програмне забезпечення, призначене для автоматизованого створення блок-схем на основі вихідного коду за один клік. Програма дає змогу використовувати блок-схеми для аналізу та оцінювання процесів рефакторингу. Наявність коментарів у блок-схемах полегшує розуміння програмного коду, сприяє перевірці коректності логіки функцій і допомагає виявляти можливі помилки. Результати можна експортувати у формати .bmp і .jpg, а також у вигляді XML-файлів для Microsoft Visio. У параметрах передбачено режими відображення блок-схем: «лише код», «лише коментарі» та «код із коментарями».

Програма характеризується розширеним функціоналом: підтримується оптимальний рівень деталізації алгоритмів для кількох блок-діаграм, коректно відображаються цикли та умовні конструкції з урахуванням їхньої складності. Також реалізовано автоматичну синхронізацію блок-схеми з програмним кодом у процесі його перегляду.

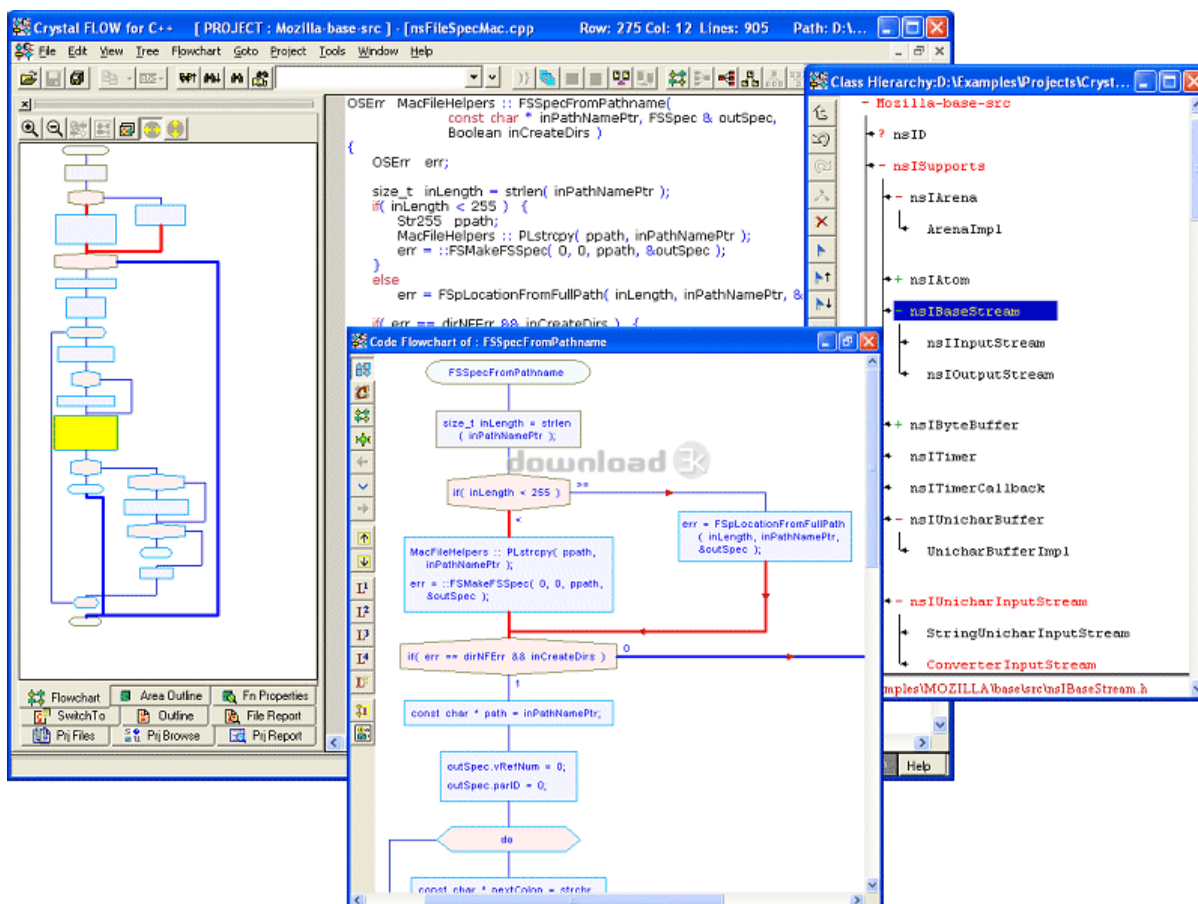


Рис. 1.5. Інтерфейс Crystal Flow

Недоліки:

- інтерфейс програми реалізовано англійською мовою;
- складність у використанні та освоєнні;
- обмежений пробний період (3 дні) для незареєстрованої версії;
- підтримка лише мов програмування C та C++.

AthTek Flowchart to Code

AthTek Flowchart to Code створено як допоміжний програмний засіб до конвертера AthTek Code to FlowChart і призначено для автоматичного перетворення блок-схем у програмний код. Інструмент дає змогу розробникам уникнути ручного введення коду рядок за рядком: достатньо побудувати блок-схему алгоритму, після чого вихідний код формується автоматично.

Використання AthTek Flowchart to Code суттєво скорочує цикл розроблення програмного забезпечення та зменшує обсяг рутинної, повторюваної роботи для інженерів-програмістів.

Програма підтримує генерацію коду кількома мовами програмування, зокрема C, C++, C#, Java, JavaScript і Delphi. Крім того, вона забезпечує можливість експорту блок-схем у формати MS Word, Visio, SVG, BMP, а також їх друк. Формування вихідного коду здійснюється одним натисканням кнопки.

AthTek Flowchart to Code позиціонується як професійний інструмент для керування проєктами у сфері програмної інженерії та вирізняється високою якістю візуального представлення діаграм серед подібних засобів. За результатами пошуку Google програма вважається одним із найпопулярніших конвертерів блок-схем у програмний код.

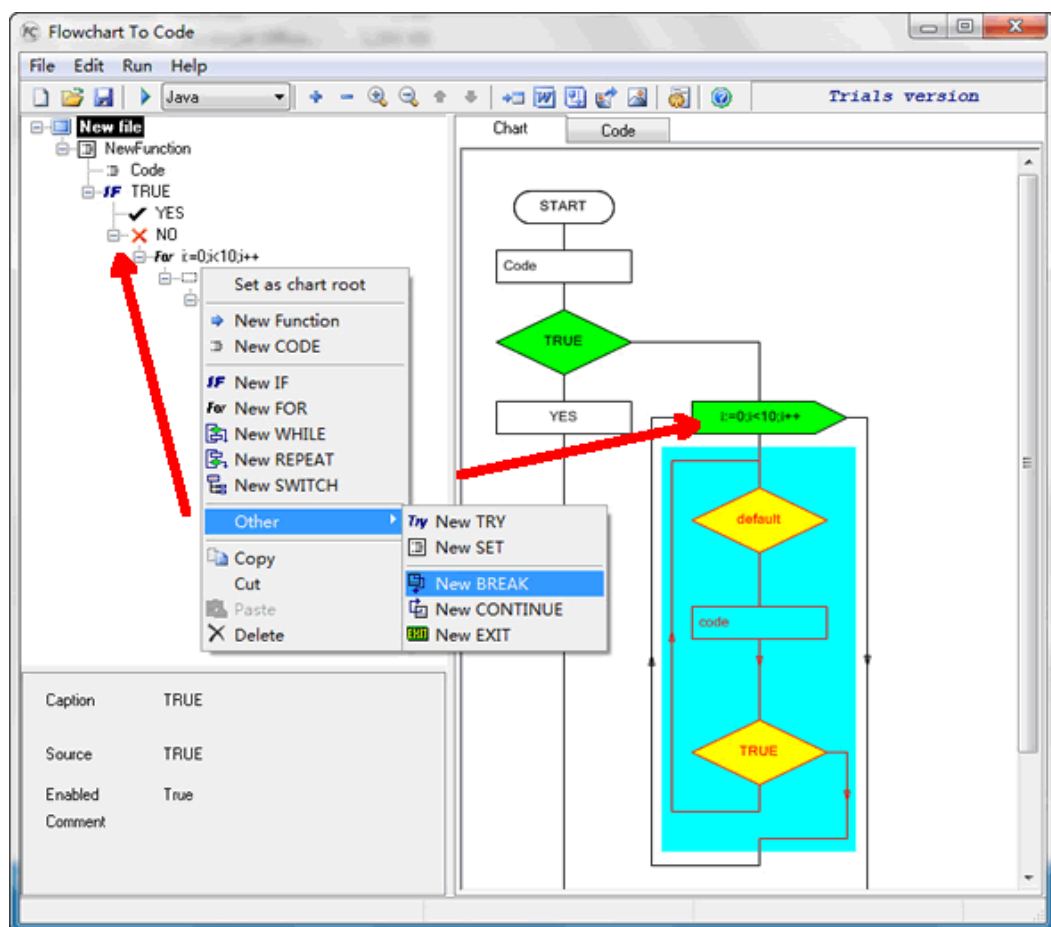


Рис. 1.6. Інтерфейс AthTek Flowchart to Code

Недоліки:

- інтерфейс доступний лише англійською мовою;
- відзначається підвищеною складністю у використанні.

Flowgorithm

Flowgorithm — це візуальне програмне середовище, яке надає можливість створювати та виконувати програми у формі блок-схем. Основна ідея

інструмента полягає в акцентуванні уваги на логіці алгоритму, а не на синтаксичних особливостях конкретної мови програмування. Побудовані діаграми можуть автоматично транслюватися у вихідний код кількома популярними мовами програмування. Розробка Flowgorithm здійснювалася в Університеті штату Каліфорнія в Сакраменто.

Програма забезпечує інтерактивне перетворення блок-схем у текстовий програмний код іншою мовою. Під час покрокового виконання алгоритму відповідні фрагменти згенерованого коду автоматично підсвічуються, що полегшує розуміння зв'язку між графічним представленням і текстовою реалізацією.

Flowgorithm підтримує такі мови програмування: C++, C#, Delphi, Java, JavaScript, Lua, Perl, PHP, Python, QBasic, Ruby, Swift 2 і 3, Visual Basic for Applications та Visual Basic .NET.

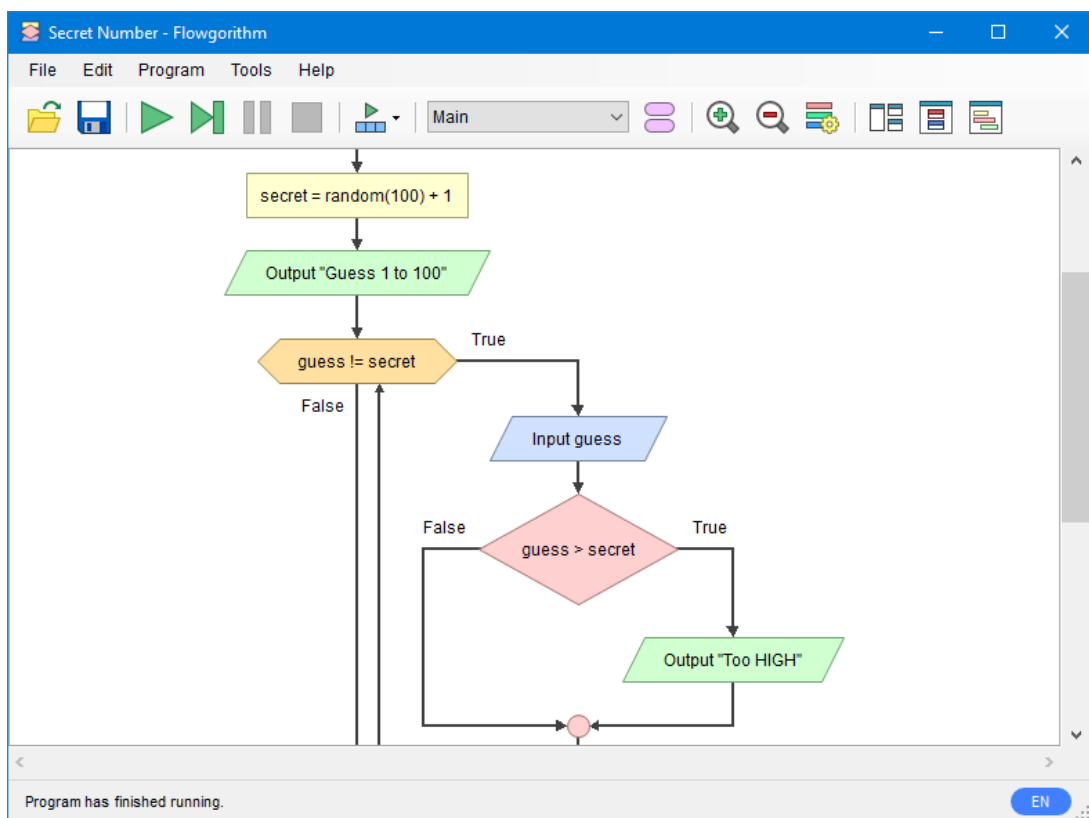


Рис. 1.7. Інтерфейс Flowgorithm

Традиційно взаємодія з програмами відбувається через консоль, де виведення здійснюється у вигляді тексту, а введення даних — з клавіатури. Такий спосіб, попри свою простоту, не завжди дозволяє чітко відрізнити введені

користувачем дані від результатів роботи програми. У Flowgorithm цю проблему вирішено шляхом використання інтерфейсу, стилізованого під вікно обміну миттєвими повідомленнями.

У режимі «chat bubbles» введення та виведення подаються у вигляді кольорових повідомлень, що відповідають елементам блок-схеми: введення користувача відображається синім кольором, а результати виконання програми — зеленим. За потреби користувач може відмовитися від цього режиму та перейти до класичного текстового відображення інформації.

Недоліки Flowgorithm:

- у режимі чат-вікна можуть зникати власні відповіді користувача;
- відсутня підтримка операторів break і continue;
- не реалізовано роботу з глобальними та/або статичними змінними;
- результати роботи конвертера іноді потребують ручного коригування;
- функції не можуть повертати масиви за допомогою оператора return.

Однак у проаналізованих програм-редакторів схем є недоліки - вони не пристосовані для українських графічних стандартів. Жодна з розглянутих програм не має функцію автоматичного розміщення блоків і не дозволяє отримати код програми відповідної побудованої блок схемою.

Проектована нами програма повинна володіти позитивними властивостями, описаними вище, а також повинна мати можливість будувати схеми за українськими стандартами, наприклад ГОСТ 19.701. Так само в створювану програму буде включено інтерпретатор програмного коду, згенерованого програмою, з його допомогою програму можна буде використовувати для вирішення алгоритмічних задач або для навчання програмуванню. Алгоритм програми можна буде редагувати в візуальному режимі, а не в режимі набору програмного коду. Потім за допомогою програми цей код можна буде виконати і перевірити результат роботи алгоритму. Це

істотно підвищить ефективність процесу навчання програмуванню на початковому етапі.

Висновки до розділу

Проведений аналіз дав змогу систематизувати розглянуті середовища відповідно до рівня їх дидактичної доцільності на різних етапах навчання алгоритмізації. Зокрема, для початкового рівня найбільш придатними є Blockly та EduBlocks; для базового й середнього етапів — Flowgorithm і draw.io; для поглибленого та методичного рівнів — Visual Paradigm і Lucidchart. Запропонований розподіл забезпечує обґрунтований вибір інструментів навчання залежно від складності навчальних завдань і поставлених педагогічних цілей.

Проектована нами програма повинна володіти позитивними властивостями, описаними вище, а також повинна мати можливість будувати схеми за стандартами. Так само в створювану програму буде включено інтерпретатор програмного коду, згенерованого програмою, з його допомогою програму можна буде використовувати для вирішення алгоритмічних задач або для навчання програмуванню. Алгоритм програми можна буде редагувати в візуальному режимі, а не в режимі набору програмного коду. Потім за допомогою програми цей код можна буде виконати і перевірити результат роботи алгоритму. Це істотно підвищить ефективність процесу навчання програмуванню на початковому етапі.

РОЗДІЛ 2.

МОДЕЛЮВАННЯ ТА РЕАЛІЗАЦІЯ РЕДАКТОРА БЛОК-СХЕМ АЛГОРИТМІВ

Проектування редактора блок-схем алгоритмів передбачає реалізацію низки послідовних етапів:

1. аналіз взаємозв'язку між елементами блок-схем і програмним кодом з метою виявлення типових конструкцій, на основі яких з певною ймовірністю можливе автоматичне формування вихідного коду;
2. встановлення відповідності кожної алгоритмічної конструкції конкретному умовному позначенню в блок-схемі;
3. добір для кожної ідентифікованої конструкції з відповідним позначенням найбільш адекватного фрагмента програмного коду;
4. перетворення сформованих моделей у синтаксичні конструкції обраної мови програмування;
5. автоматичне створення вихідного коду програми на основі отриманих синтаксичних структур.

2.1. Логічне представлення моделі поведінки програмної розробки

Поведінка програмної розробки визначається безліччю об'єктів, що обмінюються повідомленнями.

Діаграми прецедентів, один із п'яти видів діаграм в UML для моделювання динамічних аспектів системи [4], відіграють ключову роль у визначенні поведінки системи, підсистеми або класу. Кожна така діаграма відображає різноманітні прецеденти, акторів і їх взаємозв'язки.

Використання діаграм прецедентів спрямоване на моделювання системи з точки зору прецедентів або варіантів використання. Зазвичай це охоплює моделювання контексту системи, підсистеми або класу, або визначення вимог до їх поведінки.

Діаграми прецедентів є важливим інструментом для візуалізації, специфікації та документування поведінки елементів. Вони сприяють кращому розумінню систем, підсистем або класів, надаючи зовнішній огляд того, як ці елементи можуть використовуватися в конкретному контексті. Крім того, ці діаграми є важливим інструментом для тестування реалізованих систем під час прямого проектування та для розуміння їх внутрішньої структури під час зворотного проектування.

Діаграма прецедентів представлена на рисунку 2.1.

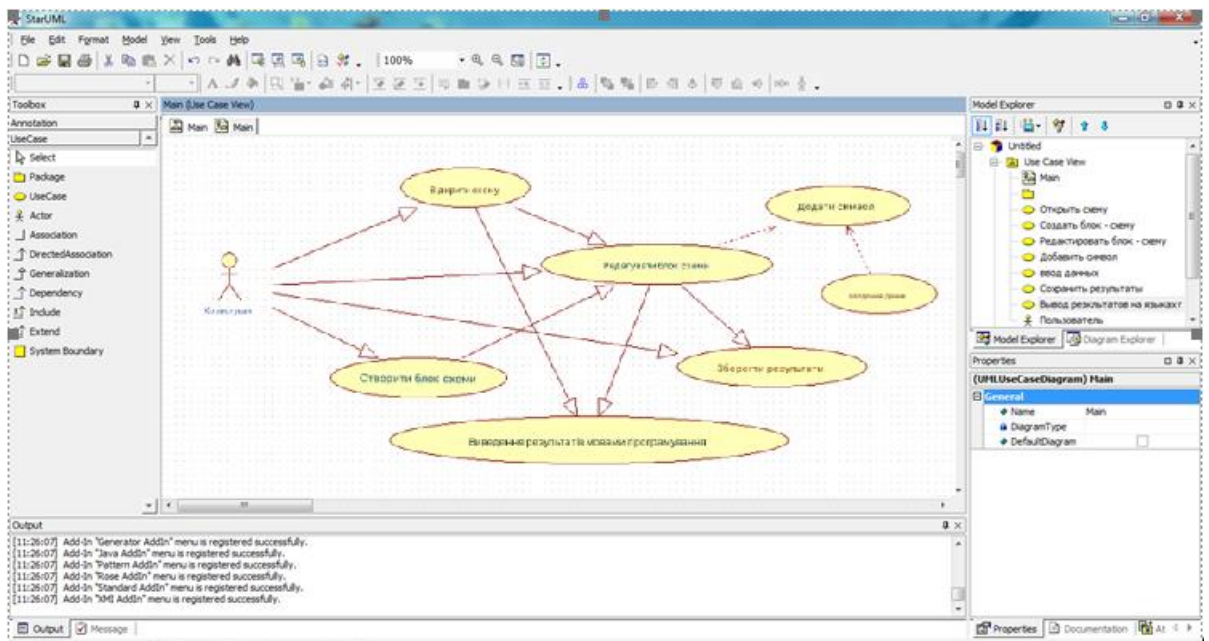


Рис. 2.1. Діаграма прецедентів

В даній системі за допомогою діаграми прецедентів були виявлені основні функції, які повинна виконувати система. Було отримано список функцій, які необхідно реалізувати в системі.

2.2. Логічне уявлення статичної моделі структури програмної розробки

Комплексний проєкт програмної системи є компіляцією моделей логічних і фізичних концепцій, які повинні бути взаємно узгоджені. В мові UML для статичного відображення системних моделей використовуються діаграми класів, які визначають структуру класів об'єктів та їх взаємозв'язки.

На діаграмі класів представлено докладний опис класів, що використовуються в програмі.

Спочатку розглянемо класи блоків. На рисунку 2.2 приведена діаграма класів блоків.

Головний блок (`ChartMainBlock`) реалізує метод збереження схеми. У програмі схема зберігається в XML формат. Для цього був обраний клас `XmlSerializer` з простору імен `System.Xml.Serialization`.

Простір імен `System.Xml.Serialization` містить класи, використовувані для серіалізації об'єктів в документи або потоки формату XML.

Центральним класом в просторі імен є клас `XmlSerializer`, який дозволяє зберігати стан об'єктів в форматі XML і потім відновлювати об'єкти з цього формату. `XmlSerializer` серіалізуються і десеріалізують об'єкти в документи XML і з них. `XmlSerializer` дозволяє контролювати спосіб кодування об'єктів в XML.

Кожен блок успадковується від абстрактного класу `Block`. Текст блоку зберігається в поле `Text`, для циклів додатково використовується поле `TextAtTheEnd` (визначає текст блоку, який закриває цикл), тип блоку визначається класом блоку. Кожен блок визначає власний метод для відтворення.

Блок певного типу реалізує тільки власний метод відтворення, все інше він успадковує від абстрактного класу `Block`.

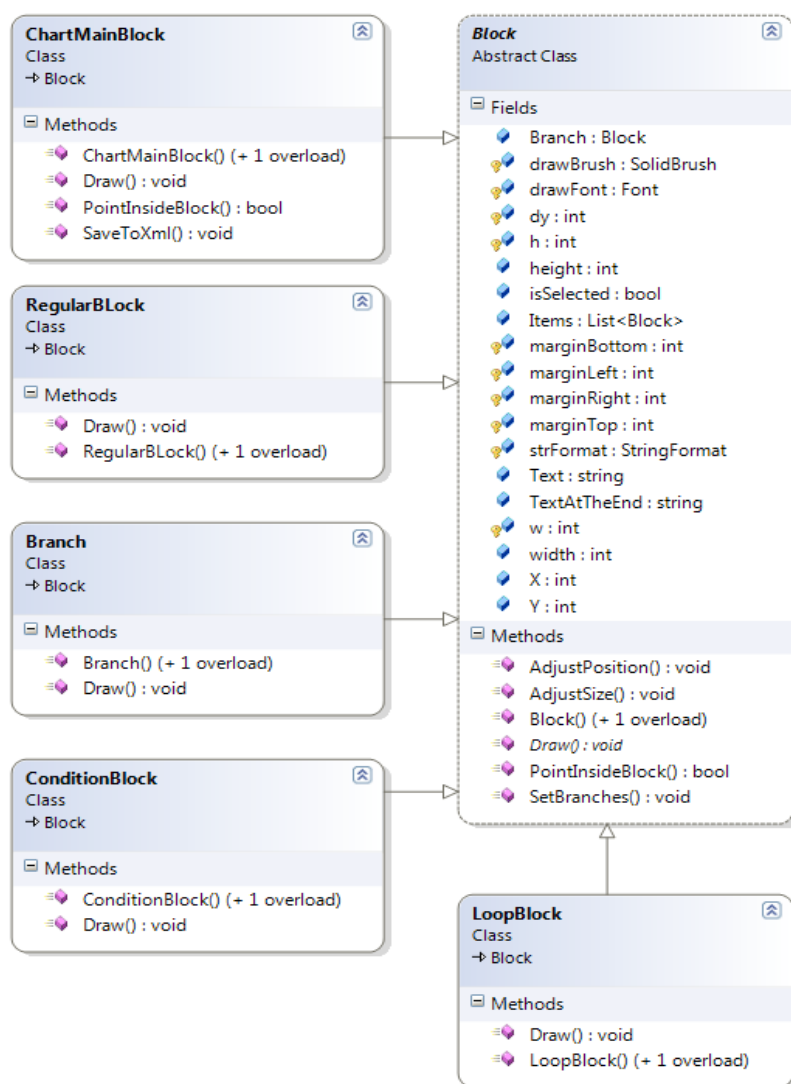


Рис. 2.2. Діаграма класів

Далі розглянемо клас, який представляє всю блок-схему. Діаграма класу блок-схеми наведена на рисунку 2.3.

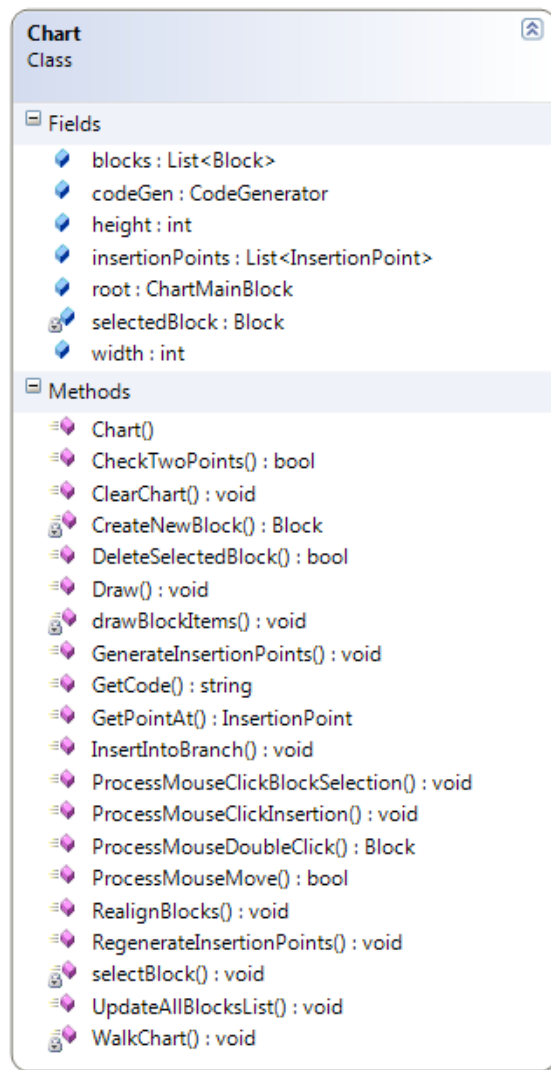


Рис. 2.3. Діаграма класу блок-схеми

Для зберігання всієї схеми в програмі є клас Chart. Він зберігає кореневий елемент схеми (блок початку схеми), список точок вставки, об'єкт для генерації коду програми зі схеми, а так само службові дані. Клас схеми реалізує методи вставки та видалення блоків, очищення всієї схеми і автоматичного вирівнювання блоків і точок вставки.

Клас блок схеми реалізує методи для автоматичного вирівнювання блоків і точок вставки (RealignBlocks, RegenerateInsertionPoints, GenerateInsertionPoints), методи для вставки і видалення блоків (InsertIntoBranch, CreateNewBlock, DeleteSelectedBlock, ClearChart), методи для обробки подій миші (ProcessMouseClickedInsertion, ProcessMouseClickedBlockSelection, ProcessMouseDownClick, ProcessMouseMove, ProcessMouseDownDoubleClick), так само цей клас має метод для отримання коду програми (GetCode).

Далі розглянемо клас точки вставки InsertionPoint. Він наведений на рисунку 2.4.

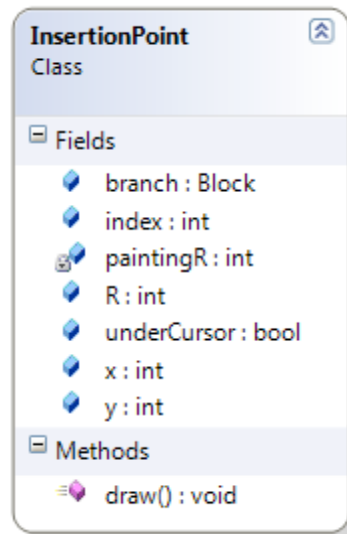


Рис. 2.4. Діаграма класу точки вставки

Далі розглянемо клас генератора коду програми. Він наведений на рисунку 2.5.

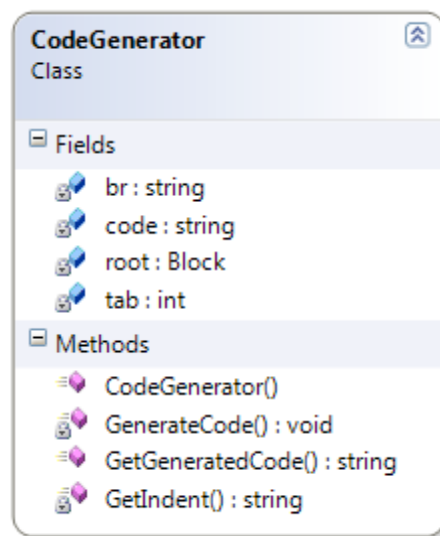


Рис. 2.5. Діаграма класу генератора коду

2.3. Математичний опис програми


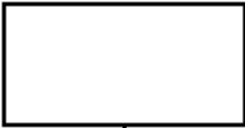
В ході реалізації виділених вище функцій необхідно описати алгоритми і моделі даних формалізованою або природною мовою.

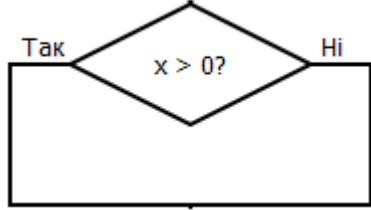
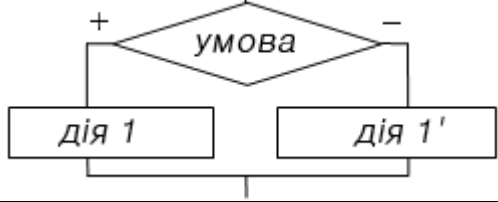
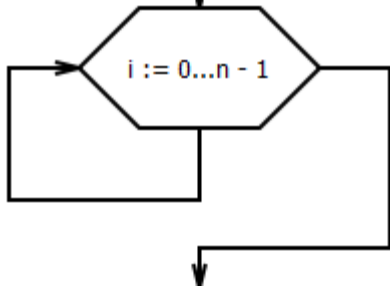
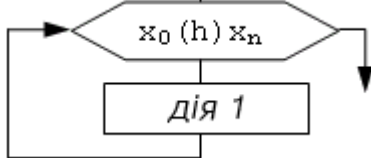
2.3.1. Опис моделі даних

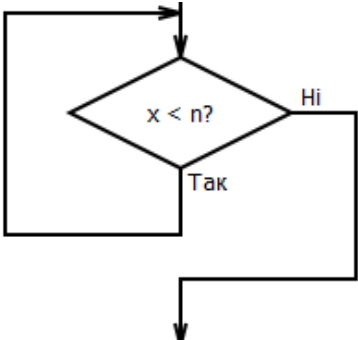
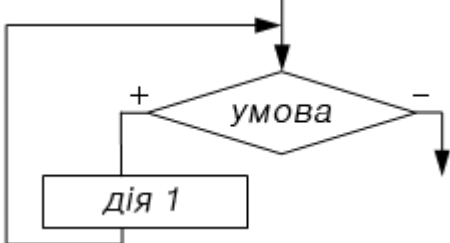
Основною структурною одиницею, що розробляється є блок. Таблиця 2.1 містить використовувані в програмі блоки.

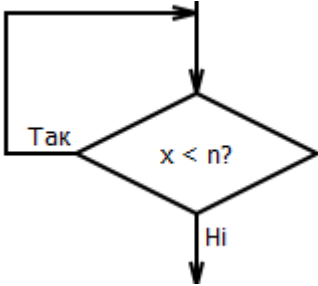
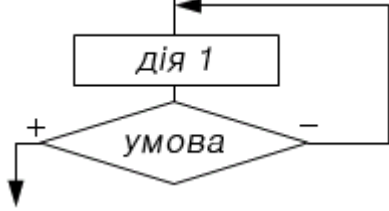
Таблиця 2.1

Основні блоки, використовувані в програмі

Найменування	Позначення	Функція
Блок введення / виведення		В інформації введення/виведення (в англ. мові часто використовується скорочення I/O - Input/Output) означає взаємодію між обробником інформації (наприклад, комп'ютером) та зовнішнім світом, який може представляти як людина, так і будь-яка інша система обробки інформації. Введення — сигнал або дані, отримані системою, а виведення — сигнал або дані, надіслані нею (або з неї).
Блок процес		Процес - виконання дії або послідовності дій, що змінюють значення, форму подання або розміщення даних.
Блок присвоювання		Присвоювання - виконання різноманітних арифметичних та логічних операцій, після яких вказані змінні приймають вказані значення. Змінна величина отримує конкретне значення лише в результаті присвоювання. Присвоювання може відбуватися в два способи: за допомогою команди присвоювання та за допомогою команди введення. Операція присвоювання хоч і виділена в окремий блок для зручності, проте вона відноситься до блоку процесу і позначається на блок-схемі як блок процесу. Формат команди присвоєння: змінна:=вираз Знак ":=" треба читати як "присвоїти".

		<p>В блок-схемах потрібно писати математичний знак рівності "=" замість ":=".</p> <p>Команда присвоювання означає наступні дії, що виконуються комп'ютером:</p> <ol style="list-style-type: none"> 1. Обчислюється вираз. 2. Змінній присвоюється отримане значення.
Блок розгалуження		<p>Розгалуження - керуюча структура, що організує виконання лише одну з двох визначених дій залежно від справедливості певної умови. Умова - питання, що має два варіанти відповіді: так чи ні.</p> 
Блок цикл з параметром		<p>Цикл з параметром, або цикл з лічильником, або арифметичний цикл - це цикл з зарання відомою кількістю повторів. В блоці модифікації вказується закон зміни параметру.</p>  <p>x_0 - початкове значення параметру; h - крок зміни параметру; x_n - останнє значення параметру. Для створення циклів з параметром необхідно використовувати правила:</p> <ol style="list-style-type: none"> 1. Параметр циклу, його початкове та кінцеве значення, а також крок повинні бути одного типу; 2. Заборонено змінювати в тілі циклу початкове, поточне або кінцеве значення параметру;

		<p>3. Заборонено входити в цикл оминаючи блок модифікації;</p> <p>4. Якщо початкове значення більше кінцевого, то крок - число від'ємне;</p> <p>5. Після виходу з циклу значення параметру невизначене та не може використовуватися в подальших обчисленнях;</p> <p>6. З циклу можна вийти не закончивши його, тоді параметр зберігає своє останнє значення.</p>
<p>Блок цикл "WHILE"</p>		<p>Цикл - керуюча структура, що організовує багаторазове виконання зазначеної дії.</p> <p>Цикл "WHILE":</p>  <p>Виконання циклу "WHILE" починається з перевірки умови, тому такий різновид циклів називають циклами з передумовою. Перехід до виконання дії здійснюється тільки в тому випадку, якщо умова виконується, в іншому випадку відбувається вихід з циклу. Можна сказати, що умова циклу "ПОКИ" - це умова входу в цикл. В окремому випадку може виявитися, що дія не виконувалася жодного разу. Умова циклу необхідно підібрати так, щоб дії, що виконуються в циклі, привели до порушення його істинності, інакше відбудеться зациклення. Зациклення - нескінченне повторення дій.</p>

<p>Блок цикл "UNTIL"</p>		<p>Цикл - керуюча структура, що організовує багаторазове виконання дії.</p> <p>Цикл "UNTIL":</p>  <p>Виконання циклу починається з виконання дії. Таким чином, тіло циклу буде реалізовано хоча б один раз. Після цього відбувається перевірка умови. Тому цикл "UNTIL" називають циклом з наступною перевіркою умови. Якщо умова не виконується, то відбувається повернення до виконання дій. Якщо умова істинна, то здійснюється вихід з циклу. Таким чином, умова циклу "ДО" - це умова виходу з циклу. Для запобігання зациклення необхідно передбачити дії, що призводять до істинності умови. Зациклення - нескінченне повторення дій.</p>
--------------------------	---	--

2.3.2. Математичний опис використовуваних моделей даних

Списки використовуються для подання кортежів.

Кортеж - це кінцева послідовність, можливо з повтореннями, елементів деякої множини E . Елементами кортежу можуть бути числа, символи деякого алфавіту, точки площині і т. д. У більш складних випадках елементами кортежу, в свою чергу, можуть бути також кортежі. Елементи, які не є кортежами, називаються атомами. Кількість елементів в кортежі називається його довжиною. Зручно розглядати кортежі, що не містять жодного елементу. Такі кортежі називаються порожніми. Довжина порожнього кортежу вважається рівною 0.

Елемент кортежу характеризується своїм номером у послідовності (кортежних номером) і змістом, тобто елементом безлічі E . Якщо довжина

кортежу дорівнює n , $n > 0$, то кортеж S зручно розглядати як відображення s безлічі $N = \{1, 2, \dots, n\}$ в безліч E . Таким чином, $s(i)$ - це i -й елемент кортежу S .

Термін "список" використовується як узагальнююча назва різних структур даних, використовуваних для подання кортежів в пам'яті комп'ютера. При поданні кортежу в пам'яті з'являється ще одна характеристика елементу кортежу - його позиція в пам'яті. У деяких випадках номер елемента в кортежі і його позиція в пам'яті пов'язані один з одним арифметичними співвідношеннями таким чином, що по номеру легко обчислюється позиція i , навпаки, по позиції обчислюється номер. В інших випадках зв'язок між номерами і позиціями задається "таблично" або здійснюється за допомогою алгоритмічних процедур. Безліч позицій позначимо через P . Іноді зручно вважати, що в безлічі P є спеціальний елемент nil , який вказує на неіснуючу область пам'яті. Таким чином, при розгляді того чи іншого списку ми маємо справу з трьома множинами E , N , P і з відображеннями на цих множинах.

Типовими при роботі зі списками є такі операції:

- знаходження позиції елемента в пам'яті по його номеру в кортежі;
- знаходження позиції елемента, наступного в кортежі за елементом з заданої позиції;
- знаходження позиції елемента, що передує в кортежі елементу із заданої позиції;
- видалення елемента, що знаходиться в заданій позиції;
- вставка в кортеж нового елемента перед елементом, розташованим в заданій позиції;
- визначення довжини кортежу.

2.3.3. Опис структур даних

Для більш глибокого розуміння завдання слід описати використовувані структури даних математично.

Нехай $L = \langle q, Listofblocks, pointer(K) \rangle$ - поточний блок,

де q - булева змінна, яка характеризує тип блоку: $q = 0$ блок є звичайним блоком, $q = 1$ -блок є блоком-гілкою.

$$\begin{cases} Listofblocks = \{ \}, \text{если } q = 1 \\ Listofblocks = \{ L_1, L_2, \dots, L_n \}, \text{если } q = 0 \end{cases}$$

де L_1, L_2, \dots, L_n - список блоків, що належить даній гілці; N-кількість блоків, що належить даній гілці; Pointer (K) - покажчик на гілку K, всередині якої знаходиться блок.

Вся схема представляється набором гілок. Головний блок зберігає в собі одну головну гілку. Блок-умова в списку вкладених блоків зберігає дві гілки, вони позначають лівий і правий виходи з блоку-умови. Усередині блоку-циклу зберігається одна гілка, яка, зберігає всі блоки, які знаходяться всередині циклу.

Точки вставки не зберігаються за допомогою блоками, вони представляються окремими об'єктами.

Нехай $T \langle L, \text{Pointer}(K) \rangle$ - точка вставки. Де L - це блок, після якого розташовується точка; Pointer (K) Покажчик на гілку, всередині якої знаходиться блок відповідний точці вставки.

2.4. Обґрунтування вибору середовища розробки системи для вирішення завдань

Qt Creator є недавно розробленим кросплатформним інтегрованим середовищем, призначеним для розробки Qt-додатків на мовах програмування, таких як C, C++, і QML. Він включає зручний відладчик та різноманітні візуальні інструменти розробки інтерфейсу, використовуючи QtWidgets і QML. Розробники цього програмного рішення ставили перед собою завдання значно спростити розробку додатків за допомогою фреймворка Qt для різних платформ.

Qt Creator - це повністю інтегроване середовище розробки (IDE), яке надає інструменти для проєктування і розробки складних додатків для безлічі настільних і мобільних платформ.



Проекти

Однією з ключових переваг Qt Creator є його здатність дозволяти розробникам працювати над проектом на різних платформах за допомогою спільних інструментів для розробки та налагодження. Проекти важливі, оскільки Qt Creator використовує їхню інформацію для компіляції та запуску додатків, подібно до вимог компілятора. Створення проекту дозволяє групувати файли, додавати власні кроки збірки, включати форми та файли ресурсів, а також встановлювати параметри для запуску додатків.

Qt Creator підтримує як створення нового проекту, так і імпорт існуючого, генеруючи необхідні файли відповідно до типу обраного проекту. Наприклад, обираючи створення додатка з графічним інтерфейсом користувача (GUI), Qt Creator автоматично генерує порожній .файл, який можна налаштувати за допомогою вбудованого Qt Designer.

Редактори

Qt Creator постачається з редактором коду і Qt Designer для проектування і створення графічних інтерфейсів користувача (GUI) з використанням віджетів Qt.

Редактор коду

Як інтегроване середовище розробки (IDE), Qt Creator володіє функціональністю, що виходить за рамки звичайного текстового редактора. Він

розуміє мови програмування C++ і QML як код, дозволяючи писати добре форматований код, автоматично доповнювати його, відображати повідомлення про помилки і попередження, надавати контекстно-залежну довідку та забезпечувати ряд інших функцій для зручності розробки.

Дизайнер інтерфейсу

Qt Designer дозволяє проектувати та налаштовувати віджети та діалоги графічного інтерфейсу користувача (GUI) з використанням віджетів Qt. Ці створені за допомогою Qt Designer елементи можуть легко інтегруватися в програмний код за допомогою механізму сигналів і слотів Qt, що полегшує визначення поведінки графічних елементів. Властивості, встановлені в Qt Designer, можуть бути динамічно змінені в коді, а також можна використовувати власні віджети та модулі для подальшого вдосконалення функціональності.

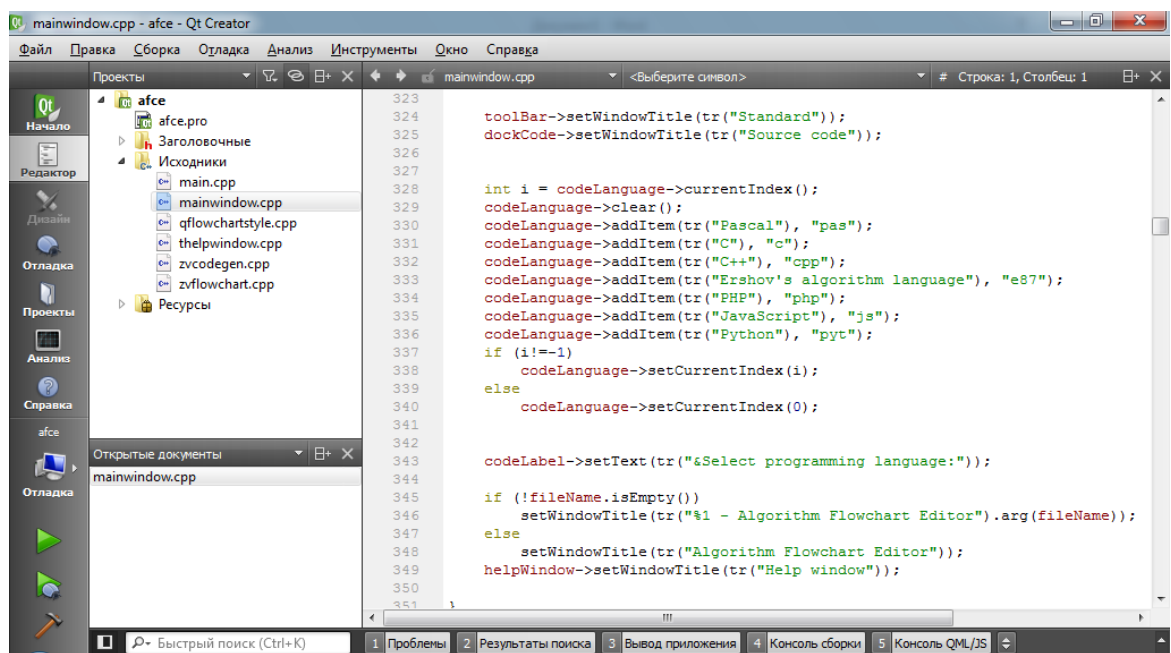


Рис. 2.6. Інтерфейс середовища розробки

Мови

Ви маєте можливість використовувати редактор для написання коду на мові Qt C++ або на мові декларативного програмування QML.

QML

Для створення дуже гнучкого інтерфейсу користувача з широким набором елементів QML ви можете використовувати мову QML. QML сприяє колаборації

між розробниками та дизайнерами, допомагаючи створювати гнучкі та користувацькі інтерфейси для портативних пристроїв, таких як мобільні телефони, медіаплеєри, нетопи і нетбуки.

QML представляє собою розширення JavaScript, що надає механізм декларативної збірки дерева об'єктів з елементів QML. Це розширення поліпшує інтеграцію між JavaScript і існуючою системою Qt, яка базується на QObject, додаючи підтримку автоматичного зв'язування властивостей і забезпечуючи мережеву прозорість на рівні мови.

Цілі

Qt Creator надає підтримку для збірки і запуску додатків на Qt на різних платформах, таких як настільні комп'ютери (Windows, Linux і Mac OS) і мобільні пристрої (Symbian, Maemo і MeeGo). Налаштування збірки дозволяють швидко перемикатися між різними цілями збірки. Коли ви збираєте додаток для мобільного пристрою, підключеного до вашого комп'ютера, Qt Creator генерує пакет установки, встановлює його на пристрої і запускає його.

Ви можете відправити пакет установки в Ovi Store. Пакети для пристроїв Symbian повинні бути підписані.

Інструменти

Qt Creator вбудований із набором корисних інструментів, таких як системи контролю версій та емулятор Qt.

Системи управління версіями

Рекомендованим способом для розробки проєкту є використання системи управління версіями. Для доступу до ваших сховищ Qt Creator використовує консольні клієнти систем управління версіями. Підтримуються наступні системи управління версіями:

- Git
- Subversion
- Perforce
- CVS
- Mercurial

Доступні функції в Qt Creator можуть варіюватися залежно від системи контролю версій. Основні можливості доступні для всіх підтримуваних систем, включаючи порівняння файлів з останньою версією у сховищі, відображення відмінностей, перегляд історії версій і деталей змін, анотацію файлів, а також можливість застосовувати та скасовувати зміни.

Емулятор Qt

Для перевірки додатків на Qt призначених для мобільних пристроїв в схожому оточенні, ви можете використовувати емулятор Qt. Ви можете змінити інформацію пристрою про його налаштуваннях і оточенні.

Емулятор Qt встановлюється як частина Nokia Qt SDK. Після установки ви можете вибрати його в якості мети збірки в Qt Creator.

Відладчики

Qt Creator може допомогти вам з налагодженням ваших додатків. Він надає інтерфейси до GNU Symbolic Debugger (gdb) і Microsoft Console Debugger (CDB) для налагодження звичайних додатків на C ++ і внутрішні відладчики для JavaScript. Це включає можливість підключити мобільні пристрої до свого комп'ютера і налагоджувати запущені на них програми.

Qt Creator відображає сиру інформацію, надану відладчиками, явним і лаконічним чином з метою спростити процес налагодження наскільки можливо без обмеження можливостей відладчиків.

2.5. Алгоритм програми

Загальні принципи створення програм через блок-схеми представлені на рисунку 2.7. Пропонована технологія не зачіпає весь процес конструювання програмного забезпечення, розглядається та частина діяльності, розробка якої не може обійтися без участі програміста: проектування та кодування.



Рис. 2.7. Технологія розробки програм

Запропонована технологія була використана для реалізації системи синтезу блок-схем окремих модулів програми, яка автоматично генерує текст програми на обраній мові програмування.

Керуючі структури. Щодо керуючих структур, відповідно до методології структурного програмування будь-яка програма може бути розглянута як структура, побудована з трьох основних типів конструкцій:

Послідовність - виконання операцій одноразово в порядку, визначеному в тексті програми.

Розгалуження - виконання однієї з двох або більше операцій, залежно від заданої умови.

Цикл - багаторазове виконання операції до тих пір, поки задовольняється певна умова.

Конструювання блок-схем модулів програми досягається шляхом вибору комбінацій конструкцій, відповідних обмеженому безлічі керуючих структур.

Затвердження. Для побудови будь-якої форми управління, яка може знадобитися при побудові блок-схеми програми, досить три базові конструкції: слідування, розгалуження і цикл.

Спочатку схема складається з двох блоків - початку і кінця. При натисканні кнопки певного блоку, програма перемикається в режим вставки. В режимі вставки всі доступні точки вставки T підсвічуються червоним кольором, а точки вставки, яка знаходиться безпосередньо під курсором синім. При кліку по точці вставки на місці цієї точки з'являється блок.

У програмі позиції блоків і точок вставки розраховуються кожен раз заново при зміні схеми. Точки вставки так само генеруються кожен раз заново при зміні схеми. Для кожної гілки за допомогою рекурсивної процедури, яка обходить всі вкладені блоки, розраховується її загальний розмір, тобто розмір з урахуванням усіх вкладених блоків. Потім блоки розміщуються на схемі, вирівнювання йде зверху вниз, зліва направо.

При видаленні блоку визначається гілка, якій належить цей блок, і зі списку елементів L_1, L_2, \dots, L_n цієї гілки видаляється відповідний блок L_i . Координати блоків схеми розраховуються заново.

Редактор надає можливість конвертувати блок-схему в вихідний текст програми для різних мов програмування, таких як Pascal, C/C++, або Алгоритмічна мова. Редактор блок-схем алгоритмів дозволяє експортувати зображення схеми в різні графічні формати: BMP, JPEG, PNG, TIFF, ICO, PPM, XBM, XPM, SVG. Програма написана на мові C++ на основі бібліотеки Qt 4.

Можливості

- генерація вихідного коду на основі блок-схеми в різні мови програмування;
- автоматичне розміщення блоків на схемі;
- експорт схеми в популярні растрові формати;
- експорт схем в векторний формат SVG;
- можливість роботи з буфером обміну;
- масштабування блок-схеми;

- підтримка декількох мов перекладів;
- конвертація блок-схеми в початковий код на кількох мовах програмування.

2.6. Опис роботи редактора блок-схем алгоритмів

Графічний інтерфейс редактора блок-схем алгоритмів представлений на рисунку 2.8.

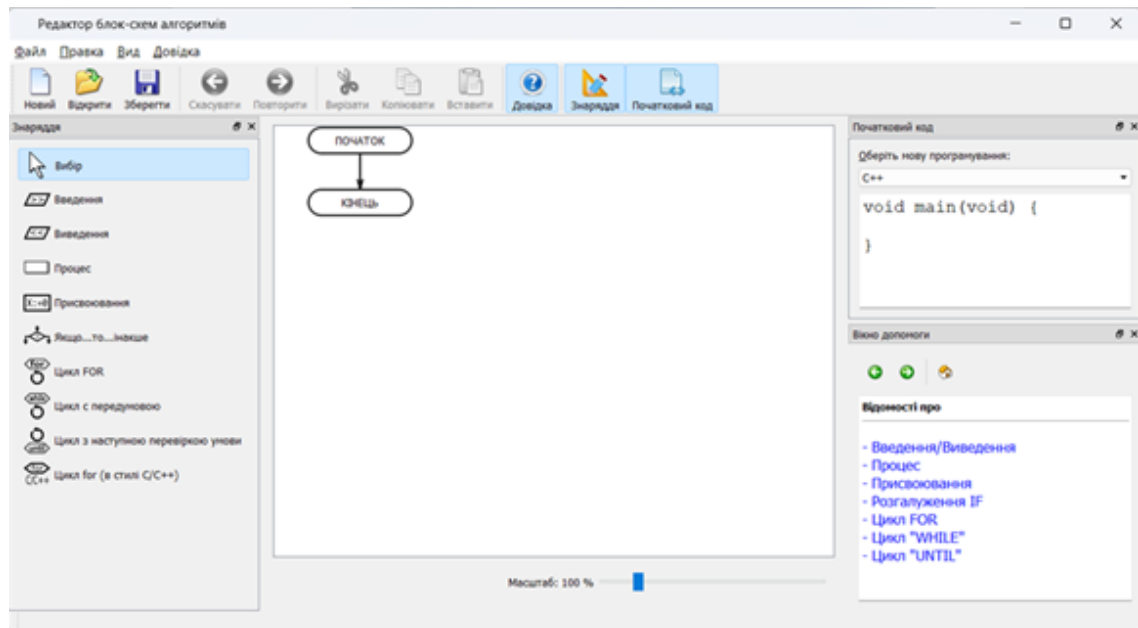


Рис. 2.8. Графічний інтерфейс програми

На головній формі програми розташовані такі компоненти, що визначають функціональність програми:

1. головне меню;
2. панель інструментів і операцій;
3. вікно з вихідним кодом програми, відповідної побудованої схемою;
4. вікно допомоги.

Головне меню програми має три розділи «Файл», «Правка» і «Довідка» програми.

Розділ «Файл» представлено на рисунку 2.9.

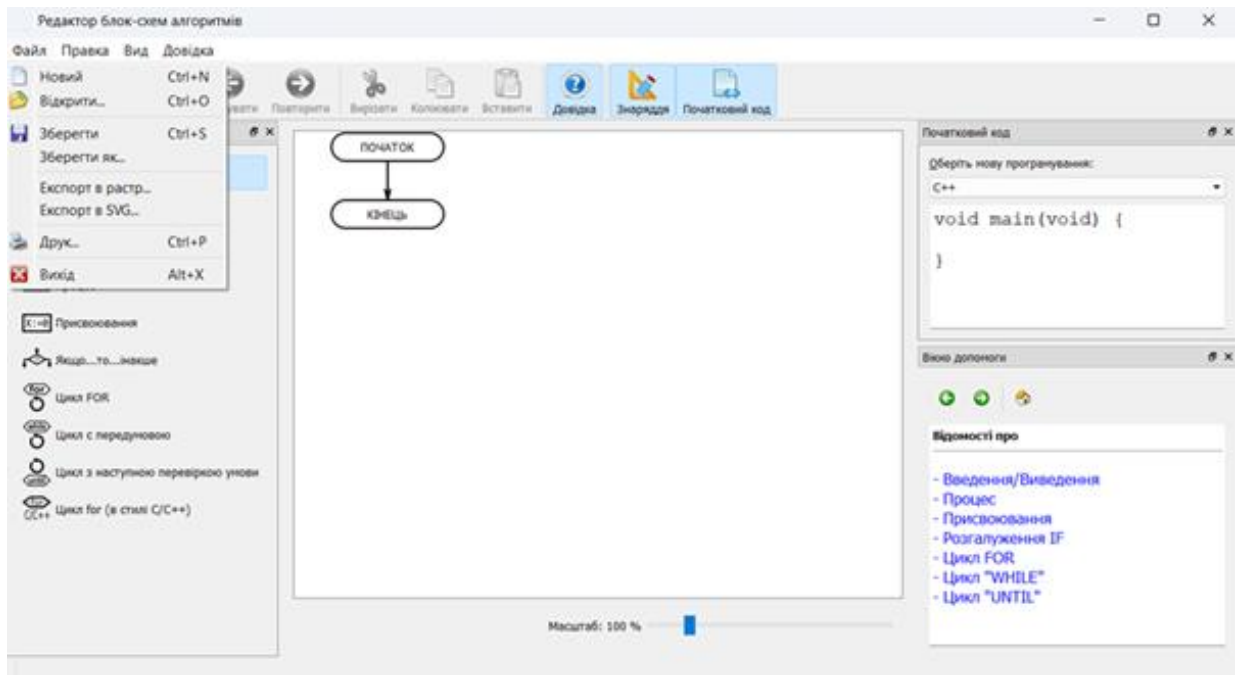


Рис. 2.9. Розділ «Файл» головного меню програми

На панелі інструментів програми розташовані блоки алгоритмічних структур рис.2.10.

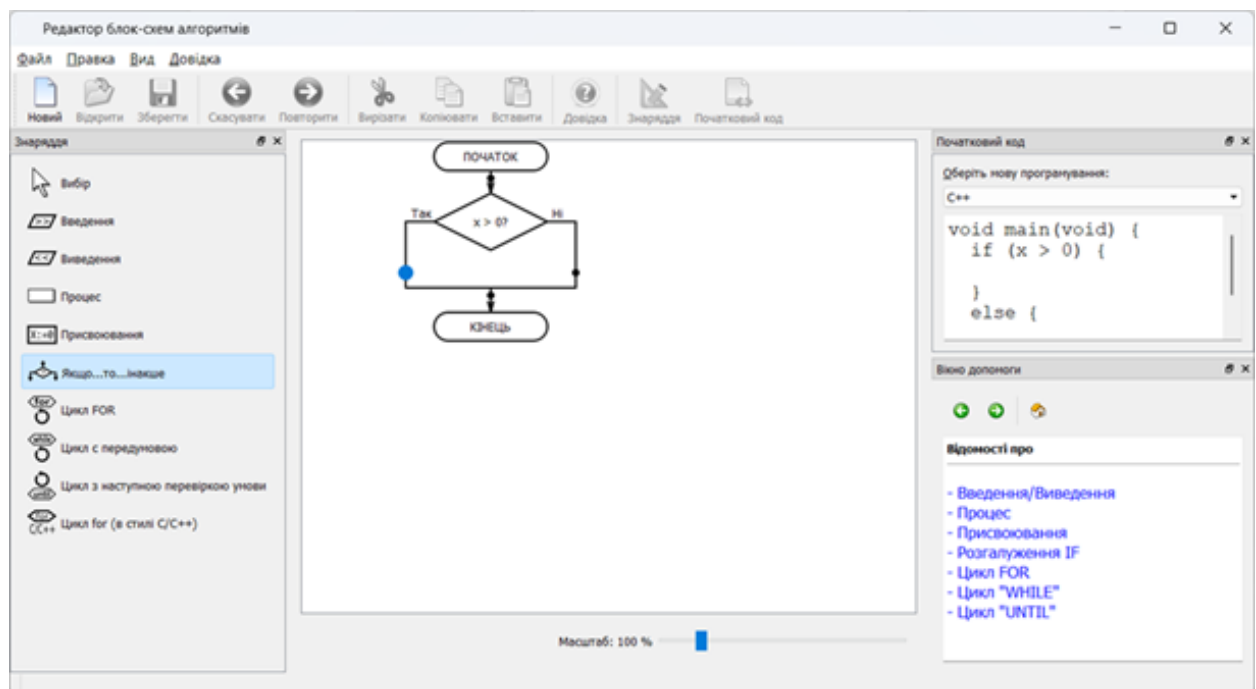


Рис. 2.10. Панель інструментів програми

Редатор надає можливість конвертувати блок-схему в вихідний текст програми для різних мов програмування, включаючи Pascal, C/C++, Алгоритмічну мову, PHP, JavaScript і Python.

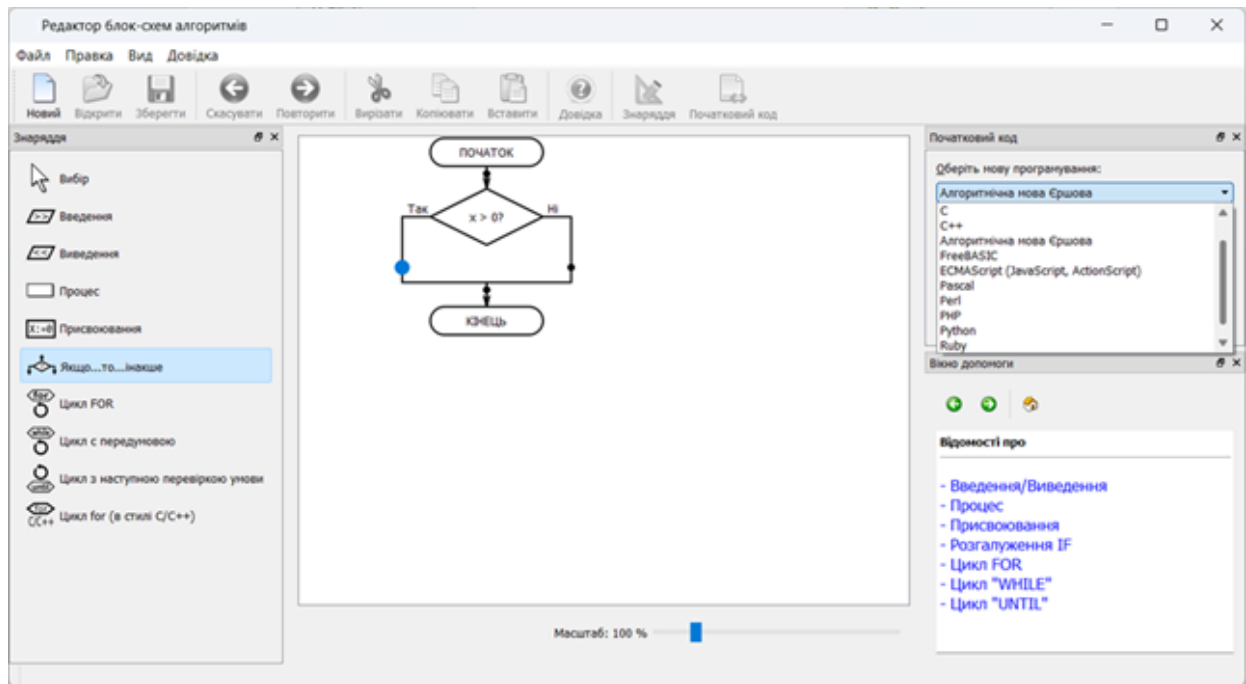


Рис. 2.11. Вибір мови програмування

Для вставки нового блоку потрібно:

1. на панелі інструментів натиснути кнопку потрібного блоку;
2. на схемі з'являться точки для вставки нового блоку (рис. 2.12).

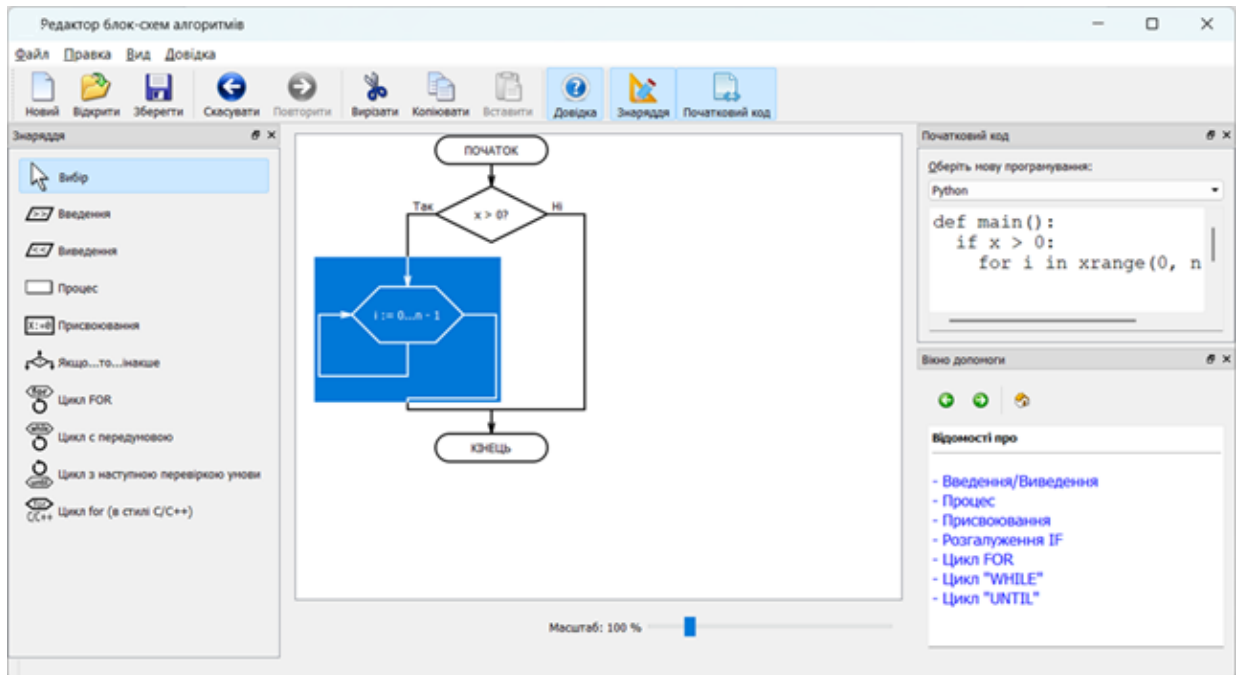


Рис. 2.12. Режим вставки нового блоку

Клікнути по потрібній точці вставки. Вставиться новий блок і програма сама вирівняє всі блоки схеми (рисунки 2.13):

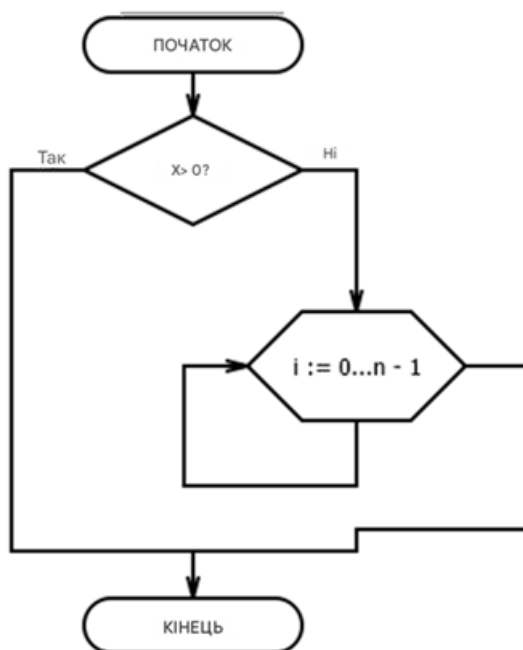


Рис. 2.13. Результат вставки нового блоку

По ходу редагування блок схеми, в правій частині вікна буде оновлюватися код схеми.

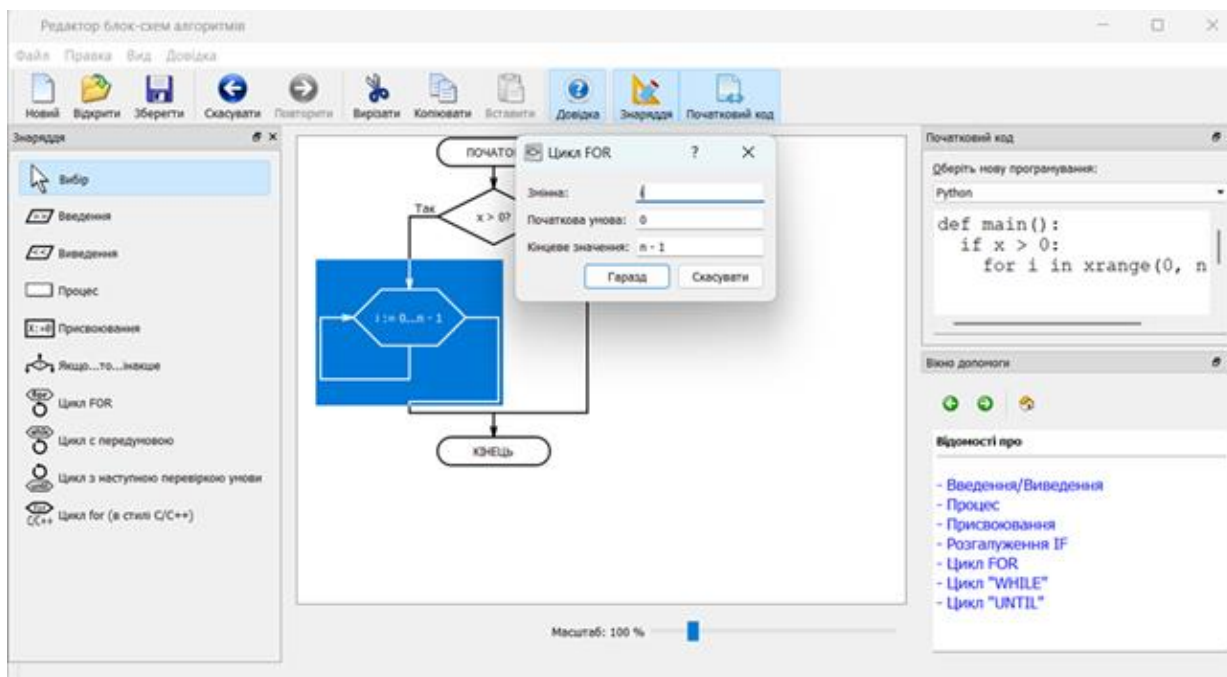


Рис. 2.14. Код схеми

Головною перевагою створеної нами програми є простота створення схем, не потрібно відкривати безліч вікон і спливаючих меню, для того, щоб створити схему. Схема створена, тепер ви можете експортувати створену схему в популярні растрові формати або в векторний формат SVG.

Висновки до розділу

Проведено аналіз процесу розробки програмного забезпечення редактора блок-схем алгоритмів. Проведено моделювання та описана архітектура розробленого додатка. Описано математичну модель системи візуального проєктування.

Створено редактор блок-схем алгоритмів. Візуальний інтерфейс редактора забезпечує можливість перетворення блок-схем у вихідний програмний код для різних мов програмування, зокрема алгоритмічної мови, Python, Pascal, C та C++. Окрім цього, редактор підтримує експорт зображень схем у поширені графічні формати: BMP, PPM, XBM, XPM, SVG, JPEG, PNG, TIFF та ICO. Програмний продукт реалізовано мовою C++ з використанням бібліотеки Qt 4.

РОЗДІЛ 3.

РОЗРОБКА ПРАКТИЧНИХ РОБІТ ЗА ТЕМОЮ «АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ» З ВИКОРИСТАННЯМ РЕДАКТОРА БЛОК-СХЕМ АЛГОРИТМІВ

3.1. Методичні підходи до використання редактора блок-схем алгоритмів у курсі інформатики

Упровадження редактора блок-схем алгоритмів у навчальний процес з інформатики відповідає сучасним вимогам Державного стандарту базової середньої освіти та концепції Нової української школи, які орієнтовані на формування в учнів ключових і предметних компетентностей, зокрема алгоритмічної та інформаційно-цифрової. Застосування такого програмного засобу сприяє реалізації компетентнісного, діяльнісного та особистісно орієнтованого підходів до навчання.

Редактор блок-схем розглядається не лише як технічний інструмент, а як дидактичний засіб, що забезпечує візуалізацію алгоритмічних конструкцій, поетапне формування вмінь аналізу задач, побудови алгоритмів і переходу від графічного подання до програмної реалізації.

Місце редактора блок-схем алгоритмів у структурі уроку інформатики

Редактор блок-схем може бути інтегрований у різні етапи уроку інформатики залежно від дидактичної мети та змісту навчального матеріалу.

На етапі актуалізації опорних знань редактор використовується для відтворення раніше вивчених алгоритмічних структур (лінійних, розгалужених, циклічних) шляхом аналізу готових блок-схем або їх фрагментів. Це дозволяє активізувати пізнавальну діяльність учнів і забезпечити наочне повторення матеріалу.

Під час пояснення нового матеріалу редактор слугує засобом демонстрації логіки побудови алгоритмів у реальному часі. Учитель поетапно створює блок-схему, коментуючи призначення кожного елемента та зв'язки між ними. Такий підхід сприяє кращому розумінню механізмів виконання алгоритмів і формуванню цілісного уявлення про їх структуру.

На етапі закріплення знань і формування практичних умінь редактор застосовується як основний інструмент діяльності учнів. Школярі самостійно або в групах розробляють блок-схеми до запропонованих задач, аналізують помилки, удосконалюють алгоритми та перевіряють їх коректність. Це забезпечує активну взаємодію з навчальним матеріалом і розвиток навичок алгоритмічного мислення.

Під час контролю та оцінювання навчальних досягнень редактор блок-схем може використовуватися для виконання практичних контрольних завдань, де оцінюється не лише правильність розв'язання задачі, а й логічність, повнота та коректність побудови алгоритму.

Форми і методи навчання з використанням редактора блок-схем алгоритмів

Використання редактора блок-схем алгоритмів у курсі інформатики дозволяє реалізувати різноманітні форми організації навчальної діяльності.

Індивідуальна форма навчання передбачає самостійну роботу учнів із редактором під час виконання практичних завдань, що сприяє розвитку самостійності, відповідальності та вмінь самоконтролю.

Парна та групова форми роботи забезпечують розвиток комунікативних умінь і навичок співпраці. Учні спільно аналізують умову задачі, обговорюють можливі способи побудови алгоритму та реалізують його у вигляді блок-схеми, аргументуючи власні рішення.

Серед методів навчання, ефективних у поєднанні з редактором блок-схем, доцільно виділити:

- **пояснювально-ілюстративний метод**, який реалізується через демонстрацію готових або частково заповнених блок-схем;
- **проблемно-пошуковий метод**, що передбачає постановку проблемної ситуації та самостійний пошук учнями алгоритмічного розв'язання;
- **практичний метод**, орієнтований на виконання навчальних і творчих завдань із використанням редактора;

- **метод проєктів**, у межах якого учні розробляють комплексні алгоритми для розв'язання прикладних задач;
- **рефлексивний метод**, що забезпечує аналіз результатів роботи, виявлення помилок і шляхів їх усунення.

Таким чином, використання редактора блок-схем алгоритмів у курсі інформатики створює умови для підвищення ефективності навчання алгоритмізації, сприяє формуванню алгоритмічного мислення та забезпечує поетапний перехід від наочного подання алгоритмів до їх програмної реалізації.

3.2. Розробка практичної роботи за темою «Інструменти побудови алгоритму та принципи розробки блок-схем»

Тема: «Інструменти побудови алгоритму та принципи розробки блок-схем».

Мета: Визначити ключові правила створення алгоритмів за допомогою графічного представлення та здобути практичні вміння роботи з редактором блок-схем алгоритмів.

Обладнання: комп'ютер типу Intel Core i3-i5 з ОС Windows (10, 11), редактор блок-схем алгоритмів.

Теоретичні основи:

Редактор блок-схем алгоритмів — це програмне забезпечення, призначене для створення та редагування блок-схем.

Блок-схеми забезпечують наочне подання алгоритмів, що полегшує розуміння їхньої структури та дозволяє візуально оцінити й оптимізувати логіку роботи. Важливою перевагою блок-схем є незалежність від синтаксису конкретної мови програмування: алгоритм можна розробляти, не прив'язуючись до певної мови, а згодом легко реалізувати його на C, Pascal, Java, Basic чи будь-якій іншій.

Редактор блок-схем алгоритмів є спеціалізованим інструментом, який містить необхідний набір засобів саме для побудови блок-схем. Це робить його значно зручнішим у використанні порівняно зі звичайними графічними

редакторами. Додаткові функції програми сприяють оптимізації процесу створення блок-схем і їх подальшого перетворення у процедури та функції мов програмування.

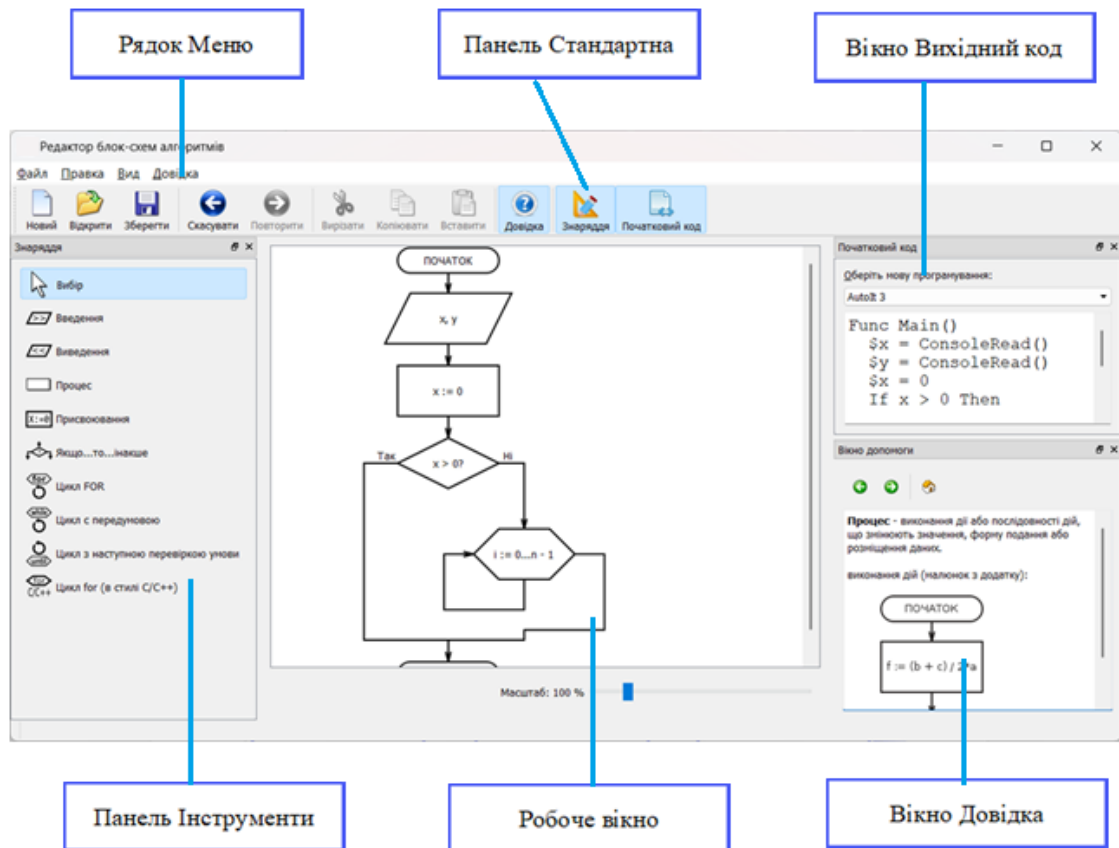


Рис. 3.1. Інтерфейс редактора блок-схем алгоритмів

Розглядаючи детальніше можливості системи, варто виділити кілька опцій, які є особливо важливими у практичній роботі.

Використання шаблонів під час створення блок-схем. Під час розробки алгоритмів часто застосовуються типові, повторювані конструкції: різні види циклів, повні та неповні розгалуження, множинні альтернативи тощо. Користувач може зберегти найбільш уживані структури як шаблони й надалі додавати їх до блок-схем одним кліком миші, що усуває потребу кожного разу створювати їх заново.

Імпорт процедур і функцій мов програмування. Редактор дозволяє завантажувати процедури та функції, написані різними мовами програмування. Це особливо корисно для аналізу алгоритмів, реалізованих мовою, з якою

користувач недостатньо добре знайомий, або для відновлення розуміння власних програм, створених раніше.

Експорт блок-схем у процедури та функції мов програмування. Ця можливість значно спрощує процес розробки програм, оскільки дозволяє автоматично перетворювати вже створену блок-схему на програмний код без необхідності ручного переписування алгоритму.

Експорт блок-схем у різні графічні формати. Завдяки цій опції блок-схеми можна використовувати в програмній документації або передавати іншим користувачам, які не працюють з даним редактором.

Порядок виконання роботи:

Завдання №1: Створення блок-схеми в редакторі блок-схем алгоритмів:

1.1. Ознайомитися з інтерфейсом вікна редактора та занотувати призначення елементів:

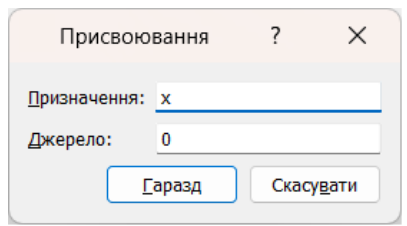
Назва елемента інтерфейсу	З яких об'єктів складається
Рядок меню	
Панель Стандартна	
Панель Інструменти	
Панель Вихідний код	
Панель Довідка	
Робоча область	

1.2. Побудувати блок-схему зображену на рисунку 3.2 «кратні пари» в редакторі блок-схем алгоритмів, для цього виконайте наступні дії:

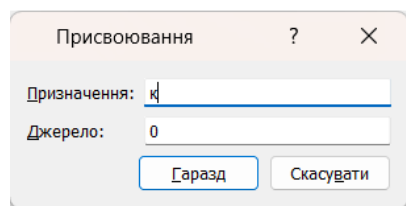
1.2.1. Виберіть перший елемент блок-схеми «присвоєння» панелі Інструменти, для цього виконайте щиглик мишею по ньому;

1.2.2. Виконайте наступний щиглик мишею по вузлу вставки символу блок-схеми (синьому кружку), який з'явиться між алгоритмічними скобками «початок/кінець» схеми Робочої області програми;

1.2.3. Виконайте подвійний щиглик мишею по символу «присвоєння», відкриється відповідний діалог –




де необхідно ввести дані по цій команді алгоритму –




1.2.4. Натисніть кнопку ОК даного діалогу;

1.2.5. Для видалення символу з блок-схеми – виділіть символ щигликом миші та натисніть [Delete];

1.2.6. Повторіть пункти 1.2.1-1.5.5 для інших символів алгоритму, що зображений на рисунку:

- «присвоєння»
(сума:=0) –  Присвоювання;
- «присвоєння»
(i:=1);
- «ЦИКЛ передумовою» (i<=n) –

-  Цикл с передумовою

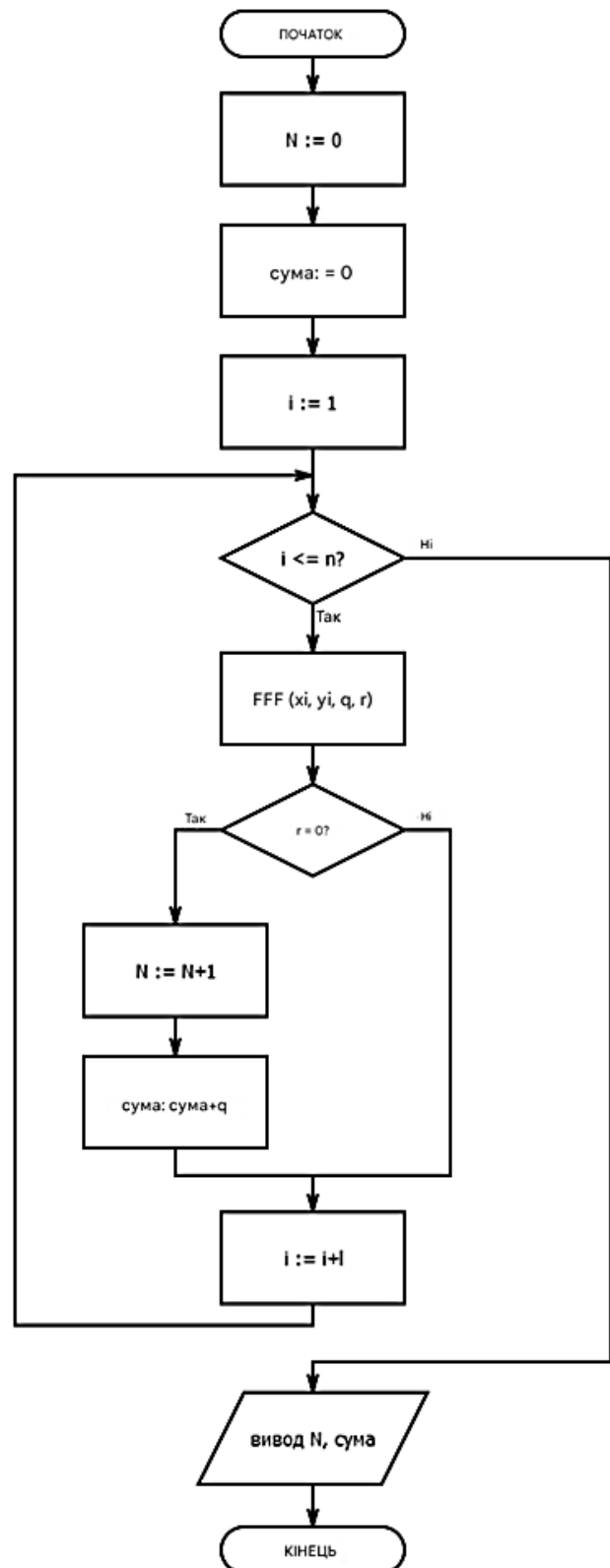
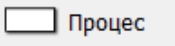
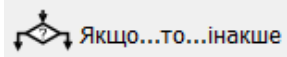
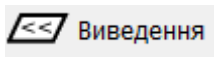
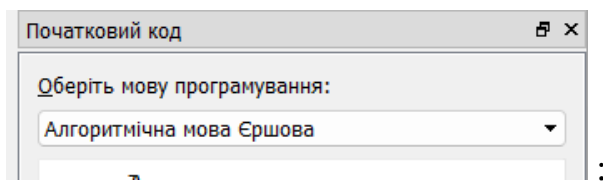


Рис. 3.2. Схема алгоритму «кратні_пари»

- В «циклі з передумовою» вставити «процес» (FFF(x_i , y_i , q , r)) –  ;
- Після символу «процес» вставити розгалуження «якщо...то...інакше» –  ;
- На гілці «Да» розгалуження вставити два символи «присвоєння» ($N:=N+1$) та ($\text{сума}:=\text{сума}+q$);
- За розгалуженням, але в циклі з передумовою вставити «присвоєння» ($i:=i+1$);
- За циклом з передумовою на гілці «Нет» вставити символ «вивід» (N , сума) –  ;

1.4. Відкрийте список *Виберіть язык програмування* панелі *Вихідний код* та виберіть пункт *Алгоритмічний язык Єршова* –



1.5. Перепишіть в звіт програму даного алгоритму на псевдокоді:

Завдання №2: Перенесення блок-схеми в текстовий документ:

- 2.1 Для перенесення зображення блок-схеми в текстовий документ необхідно задати команду меню Файл→Експорт в растр...;
- 2.2 В діалозі Зберегти як..., що відкрився. Вказати ім'я файлу типу .bmp та місце його збереження;
- 2.3 Відкрити графічний редактор Paint командою меню Пуск→Всі програми→Стандартні→Paint;
- 2.4 В редакторі Paint виконати команду Відкрити→файл, який Ви зберегли;
- 2.5 В вікні редактора з'явиться блок-схема, яку Ви зберегли. Виділіть її та виконайте команду Копіювати;

- 2.6 Відкрийте текстовий файл, куди необхідно вставити блок-схему, та виконайте команду Вставити;
- 2.7 Для перенесення тексту відповідної блок-схеми програми:
- виділіть текст програми, виконавши команду Select All контекстного меню;
 - виконайте команду Сору цього ж контекстного меню;
 - перейдіть в текстовий документ та виконайте команду Вставити.
- 2.8 Збережіть блок-схему засобами редактора блок-схем алгоритмів – виконайте команду Зберегти панелі Стандартна або меню Файл;
- 2.9 Представити викладачу текстовий файл з блок-схемою та відповідним псевдокодом та файл редактору блок-схем типу *.afc:

Позначка про виконання завдання:

Контрольні питання:

1. Дайте визначення поняттю «алгоритм».
2. Назвіть засоби представлення алгоритму.
3. Назвіть основні правила побудови блок-схем.
4. Назвіть базові алгоритмічні конструкції.
5. Назвіть призначення редактора блок-схем алгоритмів.
6. Назвіть можливості редактора блок-схем алгоритмів.
7. Як додати символ до блок-схеми?
8. Як ввести команду символу блок-схеми?
9. Як зберегти блок-схему для подальшої роботи з ним в редакторі?
10. Як зберегти блок-схему в текстовому документі?
11. Як по блок-схемі побудувати програму на псевдокодi засобами редактора?
12. Назвіть елементи з яких складається блок-схема.
13. Назвіть призначення елементів панелі Інструменти.
14. Назвіть основні файлові операції редактору блок-схем алгоритмів.
15. Назвіть основні операції редагування редактору блок-схем алгоритмів.

Висновок:

3.3. Розробка практичної роботи за темою «Базова структура алгоритму – лінійна»

Тема: «Базова структура алгоритму – лінійна».

Мета: Встановити базові принципи побудови алгоритмів лінійної структури в графічній формі та отримати практичний досвід роботи з редактором блок-схем алгоритмів.

Обладнання: комп'ютер типу Intel Core i3-i5 з ОС Windows (10, 11), редактор блок-схем алгоритмів.

Теоретичні основи:

Базова лінійна структура

Ця команда формується як послідовність дій, що виконуються одна за одною. У псевдокоді окремі команди відділяються між собою крапкою з комою. Під час виконання алгоритму дії реалізуються у тій самій послідовності, у якій вони записані. Для позначення початку та завершення команди проходження використовуються службові слова **початок** і **кінець**, які виконують роль алгоритмічних дужок.

У загальному вигляді команда проходження має такий вигляд:

початок

«дія»;

«дія»;

...

«дія»

кінець

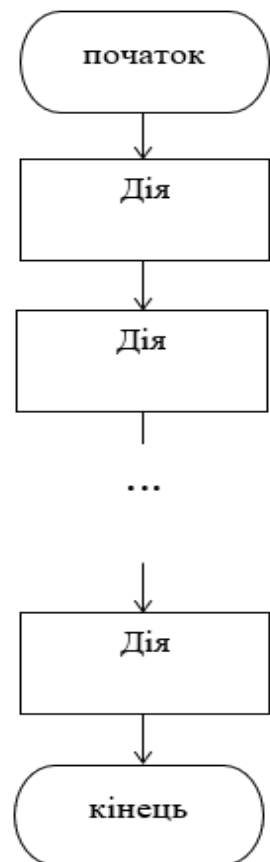


Рис. 3.3 - Схема лінійної структури

Під дією розуміють просту або складену команду. Запис таких команд може здійснюватися як в один рядок, так і у вигляді стовпчика — одна під одною. Надалі застосовуватиметься саме вертикальний спосіб запису.

Службові слова **початок** і **кінець** виконують функцію дужок, що дозволяє розглядати команду проходження як єдину дію, яка складається з послідовності простіших кроків.

Команда проходження (псевдокоди):

початок

уведення (x);

$$y = x^2 + 5;$$

$$z = \sqrt{x^2 + y^2} ;$$

кінець

Схематичне зображення команди проходження наведено на рисунку 3.3.

Саме з команд проходження формується лінійний алгоритм:

«Алгоритм називається лінійним, якщо він містить N кроків, усі кроки виконуються послідовно друг за другом від початку до кінця».

Приклад виконання завдання:

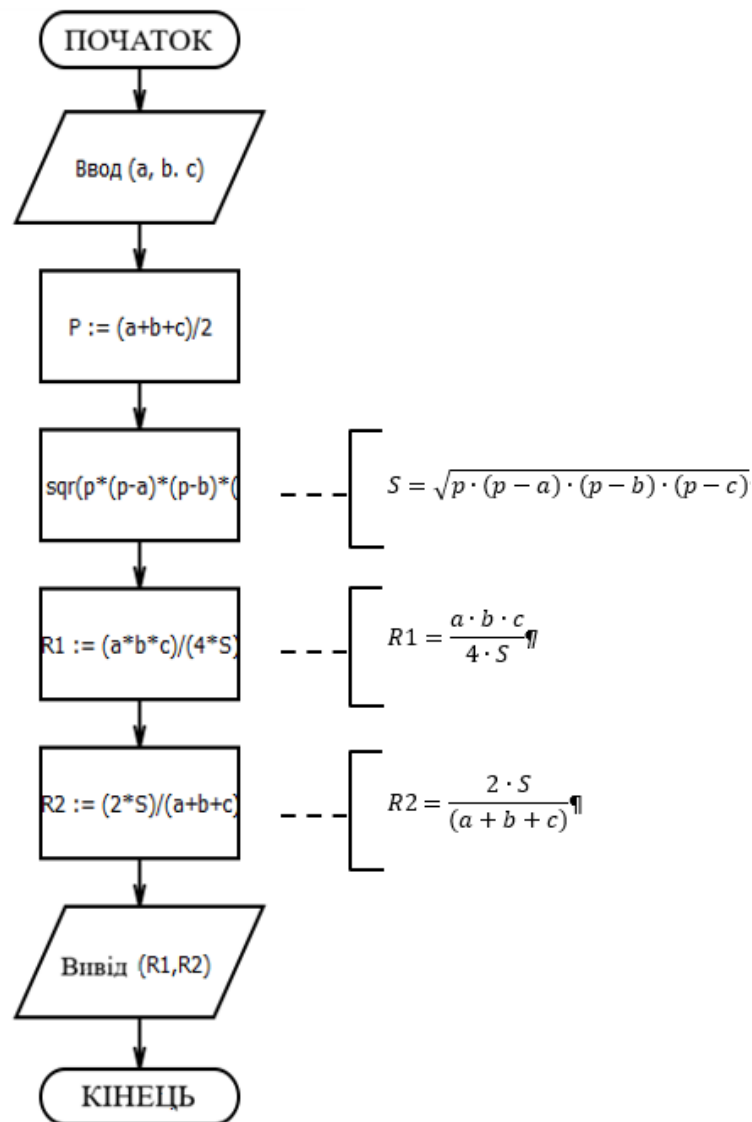
Задача: Побудувати блок-схему алгоритму обчислення радіусів описаного (R1) та вписаного (R2) кіл трикутника зі сторонами a , b , c . Здійснити виконання алгоритму для значень: $a=4,1$; $b=1,2$; $c=9,6$.

Алгоритм рішення задачі – словесний засіб:

- 1) Введення вихідних даних – a, b, c
- 2) Розрахувати полупериметр трикутника $\triangle ABC$ за формулою – $p = (a + b + c)/2$
- 3) Розрахувати площу трикутника $\triangle ABC$ за формулою Герона – $S = \sqrt{p \cdot (p - a) \cdot (p - b) \cdot (p - c)}$
- 4) Розрахувати радіус описаної навколо трикутника $\triangle ABC$ окружності – $R1 = \frac{a \cdot b \cdot c}{4 \cdot S}$

- 5) Розрахувати радіус вписаної в трикутник $\triangle ABC$ окружності – $R2 = \frac{2 \cdot S}{(a+b+c)}$
- 6) Виведення результату – R1 та R2.

Алгоритм розв'язання задачі – блок-схема:



Алгоритм розв'язання задачі – псевдокод алгоритмічною мовою:

Алг Алгоритм обчислення радіусів окружностей $\triangle ABC$

нач

|Ввід(a,b,c)

|P:=(a+b+c)/2

|S:=sqr(p*(p-a)*(p-b)*(p-c))

|R1:=(a*b*c)/(4*S)

|R2:=(2*S)/(a+b+c)

|Вивід (R1,R2)

кон

Порядок виконання роботи:

- Скласти алгоритм розв'язання задачі словесним способом і подати його в звіті;
- Створити блок-схему за допомогою редактора блок-схем алгоритмів;
- Заповнити елементи блок-схеми відповідними командами;
- Відобразити блок-схему у звіті у вигляді рисунка;
- Описати алгоритм розв'язання задачі у формі псевдокоду;
- Продемонструвати реалізоване рішення задачі в програмі-редакторі блок-схем алгоритмів викладачу.

Завдання №1: Побудувати блок-схему алгоритму для визначення сумарного опору електричного кола, яке містить три резистори, опір яких $R_1=12$ Ом, $R_2=17$ Ом, $R_3=2,9$ Ом, з'єднані паралельно:

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок-схема: Псевдокод, поданий алгоритмічною мовою:

Відмітка про виконання завдання:

Завдання №2: Створити блок-схему алгоритму визначення площі фігури, утвореної між колом радіуса R_1 та вписаним у нього колом радіуса R_2 :

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок-схема: **Псевдокод, поданий алгоритмічною мовою:**

Відмітка про виконання завдання:

Завдання №3: Розробити блок-схему алгоритму обчислення об'єму усіченого конуса з площами основ S_1 і S_2 та висотою h :

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок-схема: Псевдокод, поданий алгоритмічною мовою:

Відмітка про виконання завдання:

Контрольні питання:

1. Які існують засоби для задання алгоритмів?
2. Охарактеризуйте ці засоби.
3. Дайте визначення простої команди алгоритму.
4. Дайте визначення складної команди алгоритму.
5. Скільки існує типів базових структур алгоритмів?
6. Що таке лінійна базова алгоритмічна структура?
7. Яким символом у блок-схемі позначають команду введення або виведення даних?
8. Яким символом у блок-схемі позначають команду обчислення формули?
9. Якими символами позначають початок та кінець алгоритму?
10. Які розміри символів та їх співвідношення використовуються при побудові блок-схеми?
11. Як відбувається з'єднання символів у блок-схемі?

Висновок:

3.4. Розробка практичної роботи за темою «Базова структура алгоритму – розгалуження»

Тема: «Базова структура алгоритму – розгалуження».

Мета: встановити основні правила графічної побудови алгоритмів розгалуженої структури та набути практичних навичок роботи з редактором блок-схем алгоритмів.

Обладнання: комп'ютер типу Intel Core i3-i5 з ОС Windows (10, 11), редактор блок-схем алгоритмів.

Теоретичні основи:

Базова структура розгалуження

За допомогою команди розгалуження здійснюється вибір однієї з двох можливих дій залежно від заданої умови.

У псевдокоді ця команда в загальному вигляді подається таким чином:

якщо <умова>
то <дія 1>
інакше <дія 2>
все

Дії, що задаються після службових слів «то» та «інакше», можуть бути як простими, так і складними командами. Під час

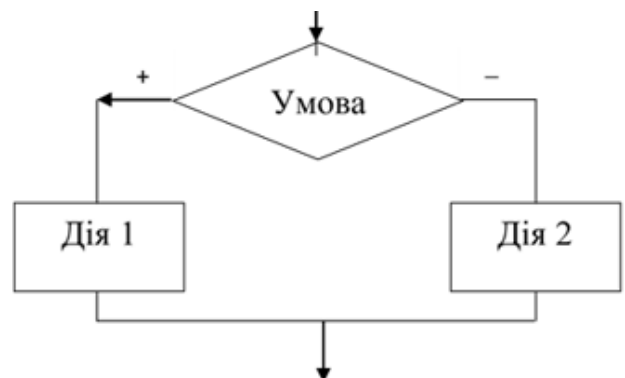


Рис. 3.4. Схема структури розгалуження

виконання команди розгалуження реалізується лише одна з дій: якщо умова виконується, здійснюється дія 1, а якщо ні — дія 2. На рисунку 3.4 наведено схему команди розгалуження. У разі виконання умови подальше виконання алгоритму відбувається за стрілкою «+», у протилежному випадку — за стрілкою «-».

Команда розгалуження може застосовуватися у скороченому вигляді (корекція), коли за умови її невиконання жодних дій не передбачається. У псевдокоді така корекція подається у вигляді:

якщо <умова>

то <дія>

все

Зображення корекції у виді схеми показане на рисунок 3.5.

На командах розгалуження будується структура алгоритму, що розгалужується.

Алгоритм вважається розгалуженим, якщо порядок виконання його кроків змінюється залежно від певних умов. Умова являє собою логічний вираз, що може набувати двох значень: «так (+)» — коли вона істинна, і «ні (-)» — коли хибна. Кожна умова складається з трьох елементів: лівої частини, знака порівняння та правої частини.

Приклади умов: $A > 0$, $X < A + Y$, $Z = 5$.

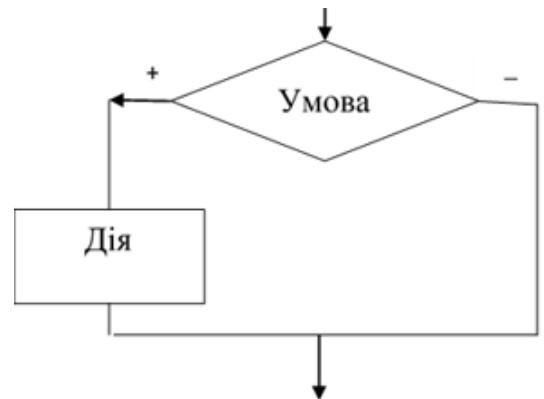


Рис. 3.5. Схема команди розгалуження в скороченій формі (корекція)

Приклад виконання завдання:

Задача: Побудувати блок-схему алгоритму для обчислення квадрата найбільшого з трьох чисел – a , b , c .

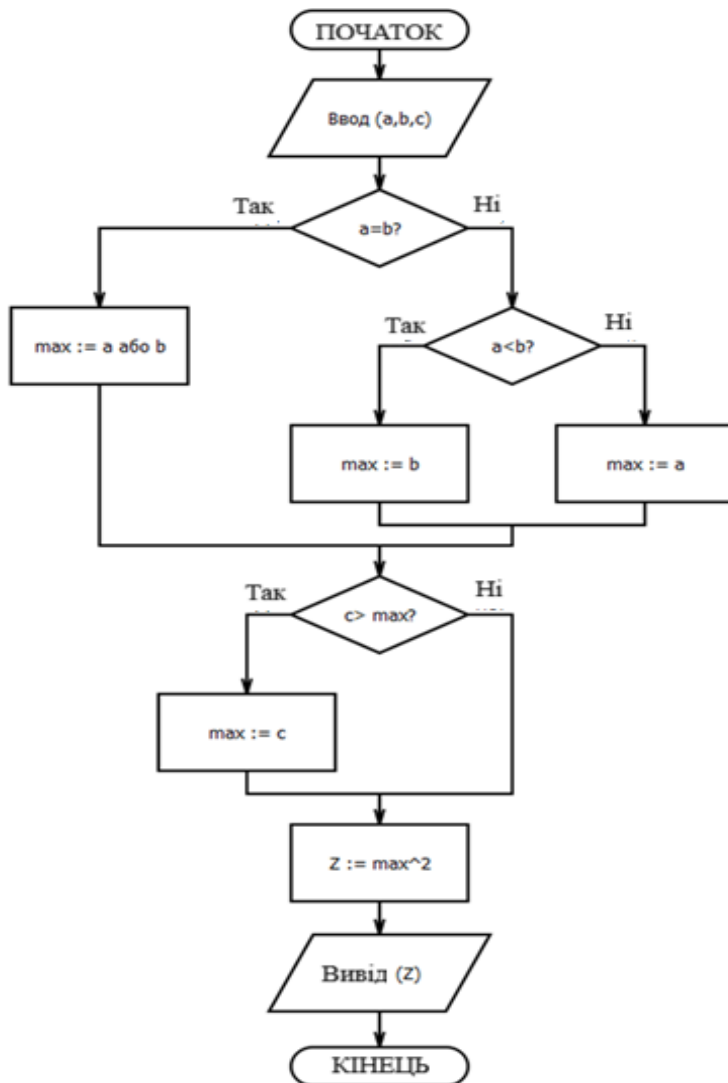
Алгоритм розв'язання задачі представлено у вигляді словесного опису:

1. Введення вхідних даних – числа a, b, c ;
2. Порівняти числа a і b , якщо $a = b$, то величині max привласнити значення числа a або b , інакше – порівняти числа a і b , якщо $a < b$, то величині max привласнити значення числа b , інакше величині max привласнити значення числа a ;
3. Порівняти числа c і max , якщо $c > max$, то величині max привласнити значення числа c інакше – пропустити – величина max має значення максимального з трьох чисел;
4. Обчислити квадрат максимального з трьох чисел Z ;

5. Вивести значення квадрату максимального з трьох чисел Z .

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок- Псевдокод, поданий алгоритмічною мовою:



алг Алгоритм
обчислення квадрату
найбільшого з трьох
чисел

```

нач
| Ввод (a,b,c)
| если a=b
| | то
| |    $\text{max} := a \text{ або } b$ 
| | иначе
| |   если a<b
| | | то
| | |    $\text{max} := b$ 
| | | иначе
| | |    $\text{max} := a$ 
| | все
| | все
| если c>max
| | то
| |    $\text{max} := c$ 
| | иначе
| | все
|  $Z := \text{max}^2$ 
| Вивод (Z)
кон
  
```

Порядок виконання роботи:

- Скласти алгоритм розв'язання задачі словесним методом та оформити його у звіті;
- Створити блок-схему за допомогою редактора блок-схем алгоритмів;
- Внести команди у відповідні елементи блок-схеми;
- Відобразити блок-схему у звіті графічно;
- Оформити псевдокод програми для розв'язання задачі;

- Продемонструвати готове рішення задачі в редакторі блок-схем алгоритмів викладачу.

Завдання №1: Скласти схему алгоритму обчислення значень функції $B(p)$, де

$$B(p) = \begin{cases} \sqrt{p} + 1,5 \cdot p, & \text{при } p \geq 1; \\ p^3 + 1, & \text{при } p < 1; \end{cases} \quad \text{а } p \text{ залежить від } x,$$

$$\text{як } p(x) = \begin{cases} 1/x^2, & \text{при } x < 1 \\ 2 \cdot x + \sin x, & \text{при } x \geq -1 \end{cases}$$

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок-схема: **Псевдокод, поданий алгоритмічною мовою:**

Відмітка про виконання завдання:

Завдання №2: Скласти блок-схему алгоритму визначення: чи є кут $\angle C$ трикутника $\triangle ABC$ зі сторонами a, b, c тупим, прямим або гострим:

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок- Псевдокод, поданий алгоритмічною мовою:

Відмітка про виконання завдання:

Завдання №3: Скласти схему алгоритму перевірки положення точки з координатами $A(x_0, y_0)$ відносно кола з радіусом r – в колі, за колом або на окружності, рівняння окружності задане формулою $r^2 = x^2 + y^2$:

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок- Псевдокод, поданий алгоритмічною мовою:

Відмітка про виконання завдання:

Контрольні питання:

1. Які типи розгалужень застосовуються в алгоритмах?
2. Опишіть особливості кожного з цих типів розгалужень.
3. Дайте визначення команді розгалуження в алгоритмі.
4. Які конструкції псевдокоду використовують для опису розгалужень?
5. Яку базову алгоритмічну структуру називають розгалуженою?
6. Яким графічним символом у блок-схемі позначають перевірку умови?
7. Що розуміють під поняттям «гілка» в алгоритмі з розгалуженням?
8. Скільки гілок може містити розгалуження в алгоритмі?
9. Дайте визначення терміну «умова» в алгоритмах розгалуження.
10. Які правила формулювання умови в алгоритмі розгалуження вам відомі?

Висновок:

3.5. Розробка практичної роботи за темою «Базова структура алгоритму – циклічна»

Тема: «Базова структура алгоритму – циклічна».

Мета: Ознайомитися з основними правилами графічного оформлення алгоритмів циклічної структури та набути практичних навичок роботи з редактором блок-схем алгоритмів.

Обладнання: комп'ютер типу Intel Core i3-i5 з ОС Windows (10, 11), редактор блок-схем алгоритмів.

Теоретичні основи:

Базова структура повторення (цикл)

У більшості алгоритмів присутні послідовності команд, що багаторазово повторюються. Якщо оформлювати такі дії у вигляді складеної команди проходження, кожену з них довелося б записувати стільки разів, скільки вона

повторюється, що є неефективним. Саме тому для опису багаторазових дій застосовують спеціальну конструкцію — цикл. Складна команда циклу, яку також називають командою повторення, містить умову, за допомогою якої визначається кількість повторень.

Будь-який циклічний алгоритм складається з таких типових елементів:

- ✓ **Задання параметрів циклу** — встановлення початкового та кінцевого значень змінної (параметра) циклу. Змінна циклу — це величина, зміна якої визначає багаторазове виконання циклу.
- ✓ **Опис тіла циклу** — сукупність дій, що багаторазово виконуються в циклі для поточного значення його параметра, починаючи з початкового.
- ✓ **Зміна кроку циклу** — формування наступного значення параметра циклу шляхом додавання до поточного значення змінної величини кроку. Крок циклу — це величина, на яку змінюється параметр під час кожного повторення.
- ✓ **Перевірка умови виконання (завершення) циклу** — якщо поточне значення параметра перевищує задане кінцеве значення, виконання циклу припиняється.

Існує три типи циклів повторення:

Цикл із відомою кількістю повторень (з параметром):

Команда повторення з параметром виконується певну кількість разів — стільки разів, скільки значень приймає параметр циклу на визначеному інтервалі від початкового до кінцевого значення, з урахуванням заданого закону зміни (кроку циклу).

Цикл із передумовою (невідоме число повторень):

Виконання такого циклу відбувається наступним чином:

- ✓ Спершу перевіряється умова виконання циклу (тому цикл називають циклом із передумовою).
- ✓ Якщо умова дотримується, виконується тіло циклу.
- ✓ Після виконання тіла циклу умова перевіряється знову.
- ✓ Цикл продовжується доти, доки умова залишається істинною.

Для правильного функціонування такого циклу команда в тілі циклу повинна впливати на значення умови.

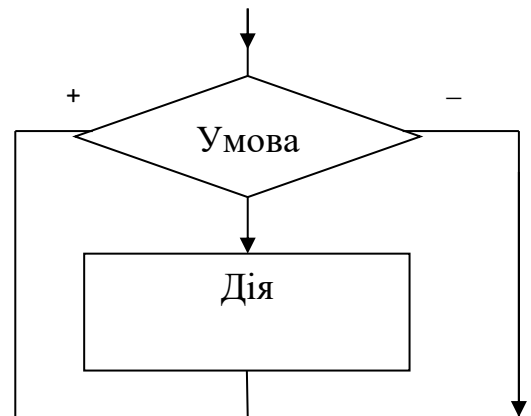


Рис. 3.6. Схема циклу повторення з перед умовою

Цикл із після умовою:

Принцип роботи схожий на цикл з передумовою, але перевірка умови відбувається після виконання тіла циклу. Повторення триває до тих пір, поки умова не стане істинною, тобто тіло циклу виконується щонайменше один раз.

Запис циклу з передумовою в псевдокоді виглядає так:

поки <умова>

повторювати <дія>

Під дією розуміється проста або складена команда. Виконання циклу з передумовою відбувається так: спочатку перевіряється умова (тому цикл і називають циклом із передумовою), і якщо вона істинна, виконується команда, що розташована після службового слова **повторювати**. Потім умова перевіряється знову, і цикл триває доти, доки умова залишається виконаною. Для коректної роботи циклу необхідно, щоб команда в його тілі впливала на значення умови.

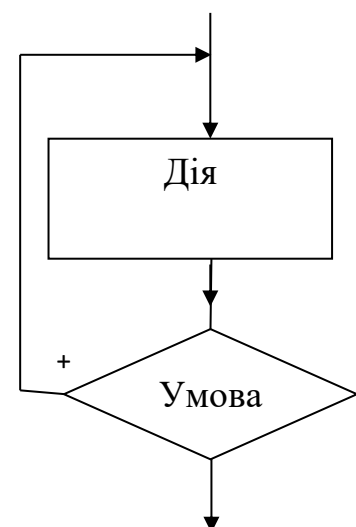


Рис. 3.7. Схема циклу повторення з післяумовою

Запис циклу з передумовою у вигляді блок-схеми представлений на рисунку 3.6.

Команда повторення з після умовою працює подібно до циклу з передумовою, проте умова перевіряється після виконання команди. Виконання команди повторюється доти, поки умова не стане істинною, тобто повторення відбувається **до виконання умови** (з цієї причини такий цикл також називають циклом «до»).

У псевдокоді цикл із після умовою записується у вигляді:

повторювати <дія>

до <умова>

Схема цього циклу наведена на рисунку 3.7.

На відміну від попередніх циклів, **команда повторення з параметром** виконується певну кількість разів — стільки, скільки значень приймає параметр на заданому інтервалі, від початкового до кінцевого значення, з урахуванням визначеного закону зміни (кроку циклу).

Схема цього циклу наведена на рисунку 3.8.

Саме з команд повторення формуються **циклічні алгоритми**.

Алгоритм називається циклічним, якщо визначена послідовність кроків виконується кілька разів у залежності від заданої величини. Ця величина називається параметром циклу.

Приклади виконання завдання:

Задача №1: Скласти блок-схему алгоритму обчислення суми членів ряду

$$s = \sum_{i=1}^{\infty} \frac{1}{(2i-1)(2i+1)}$$

з точністю до члена ряду меншого $E=10^{-3}$.

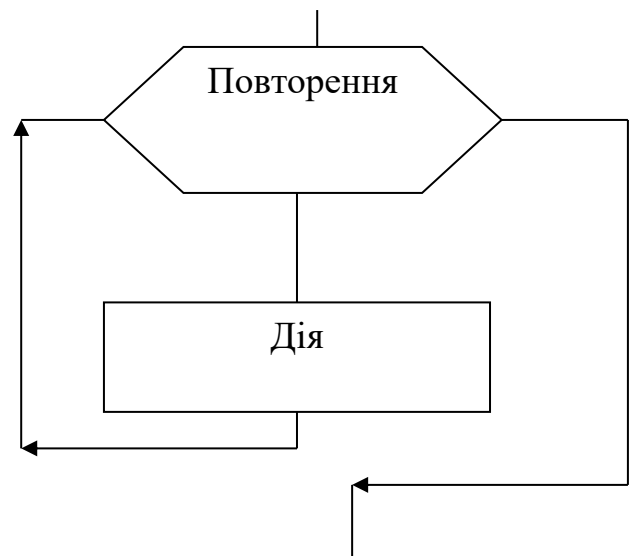


Рис. 3.8. Схема циклу повторення з параметром

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

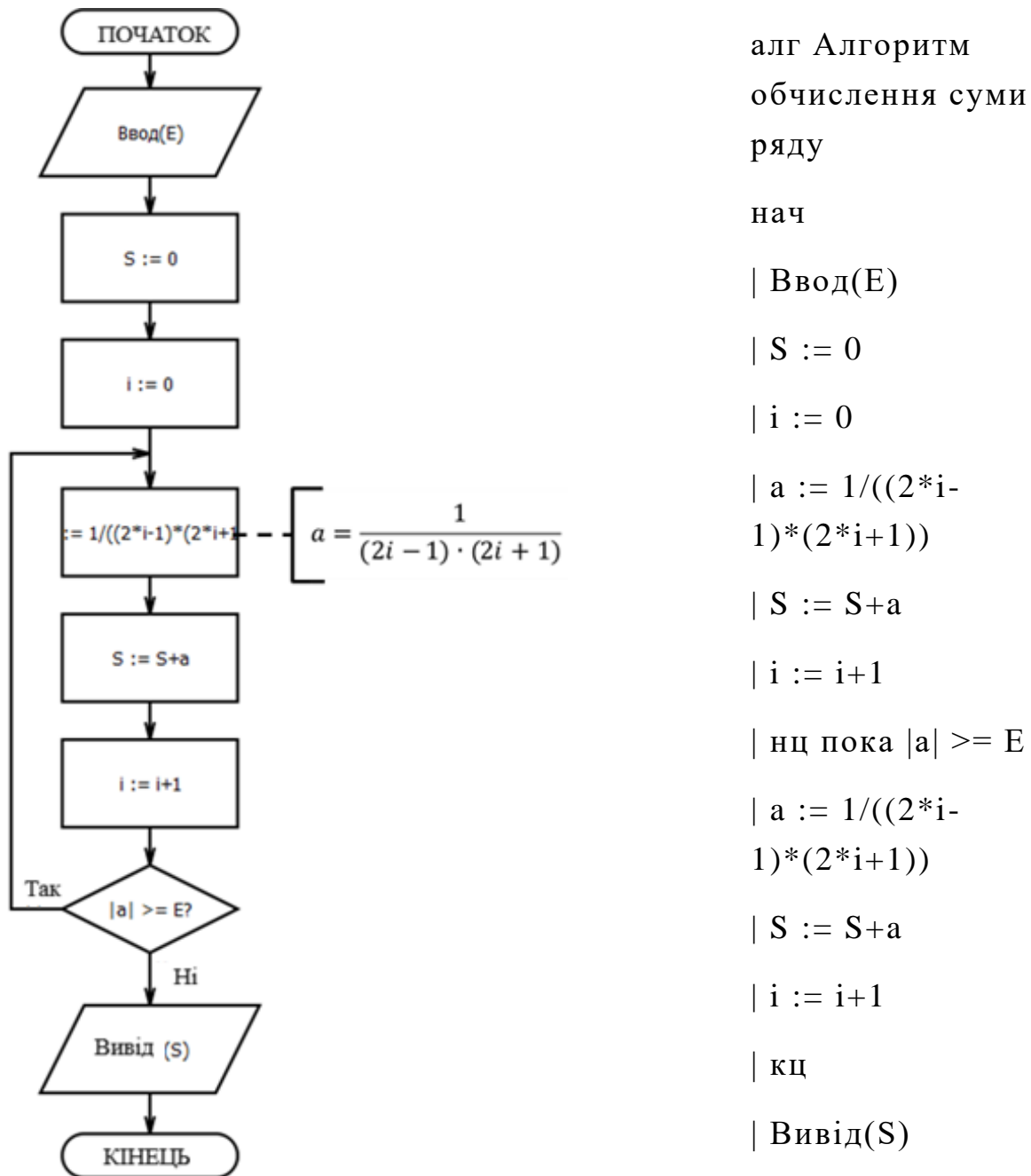
- 1) Введення вихідних даних – число E ;
- 2) Присвоїти величині $S=0$, а $i=1$;
- 3) Присвоїти a значення, що розраховується за формулою :

$$a = \frac{1}{(2i - 1) \cdot (2i + 1)} ;$$

- 4) Присвоїти величині S її попереднє значення збільшене на значення величини a – сума попереднього значення S та поточного члену ряду a ;
- 5) Обчислити наступне значення i – збільшити на 1;
- 6) Перевірити умову розрахунку суми ряду – $|a| \geq E$ – якщо умова виконується, то повторити дії пунктів 3-6, інакше вивести значення суми ряду S .

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок-Псевдокод, поданий алгоритмічною мовою:



Задача №2: Скласти блок-схему алгоритму визначення значень функції $y=x + \sin(x)$ на відрізку $[-2;2]$ із кроком 0.25.

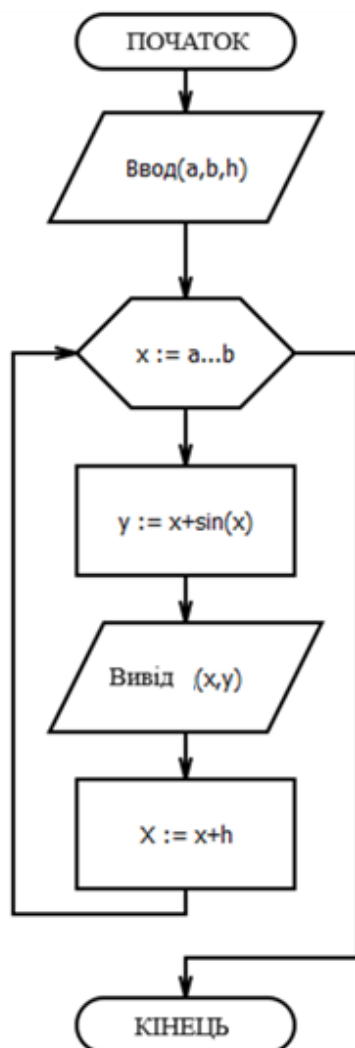
Алгоритм розв'язання задачі представлено у вигляді словесного опису:

- 1) Введення вихідних даних – числа a, b, h;
- 2) Завдання параметрів циклу, де x - змінна циклу, a – початкове значення змінної циклу x, b – кінцеве значення змінної циклу x;

- 3) Перевірка умови роботи циклу – якщо поточне значення $x \geq b$, то цикл працює ще один прогін, інакше цикл припиняє роботу і виконує перехід до пункту 7);
- 4) Розрахувати поточне значення величини y для поточного значення x за формулою: $y = x + \sin(x)$;
- 5) Виведення поточного значення x та відповідного йому значення y ;
- 6) Визначення наступного значення змінної циклу x шляхом нарощування величини кроку циклу h до поточного значення x ;
Перехід до виконання пункту 3);
- 7) Кінець алгоритму.

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок- Псевдокод, поданий алгоритмічною мовою:



алг Алгоритм

нач

| Ввод(a,b,h)

| нц для x от a до b

| | y := x+sin(x)

| | Вивод(x,y)

| | X := x+h

| кц

кон

Порядок виконання роботи:

1. Скласти алгоритм розв'язання задачі словесним способом та оформити його у звіті;
2. Створити блок-схему за допомогою редактора блок-схем алгоритмів;
3. Внести команди у відповідні елементи блок-схеми;
4. Відобразити блок-схему у звіті графічно;
5. Описати алгоритм розв'язання задачі у вигляді псевдокоду;
6. Продемонструвати реалізоване рішення задачі викладачу в програмі-редакторі блок-схем.

Завдання №1: Скласти схему алгоритму визначення ступеню дійсного числа a - $y = a^n$, з натуральним показником n .

Для обчислення використовуйте формулу $a^n = \underbrace{a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot \dots \cdot a}_{n \text{ разів}}$

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок-схема: Псевдокод, поданий алгоритмічною мовою:

Відмітка про виконання завдання:

Завдання №2: Скласти схему алгоритму обчислення:

$$N = \frac{15}{x} + \frac{14}{x+1} + \frac{13}{x+2} + \dots + \frac{1}{x+14} :$$

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок- Псевдокод, поданий алгоритмічною мовою:

Відмітка про виконання завдання:

Завдання №3: Скласти схему алгоритму визначення значень функції $y=F(x)$ на відрізку $[A,B]$ з кроком H , де $F(x)=\sin^2x+1$, $A=1,2$; $B=3,6$; $H=0,2$:

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок- Псевдокод, поданий алгоритмічною мовою:

схема:

мовою:

Відмітка про виконання завдання:

Завдання №4: Скласти схему алгоритму обчислення суми членів ряду: $S = \sum_{i=1}^n F_1 / F_2$, де $F_1 = i^2 \cos^i x$, $F_2 = i^2 + 2i + 3$, $x = 2,6$, $N = 24$:

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок- Псевдокод, поданий алгоритмічною
схема: мовою:

Відмітка про виконання завдання:

Контрольні питання:

1. Як визначити поняття «цикл»?
2. Які типи циклів існують в алгоритмах?
3. Опишіть особливості кожного з цих типів циклів.
4. Що таке команда повторення в алгоритмі?
5. Які конструкції псевдокоду застосовують для опису циклів?
6. Яку базову алгоритмічну структуру відносять до циклічних?
7. Яким символом на блок-схемі позначають команду повторення?
8. Які стандартні елементи входять до складу циклу?
9. Що розуміють під «тілом циклу» в циклічному алгоритмі?
10. Що означає «крок циклу» в циклічному алгоритмі?
11. Як визначається «змінна циклу» в циклічному алгоритмі?
12. Що таке «нарощування кроку циклу» і яку функцію воно виконує?

Висновок:

3.6. Розробка практичної роботи за темою «Основні алгоритми для обчислення суми та підрахунку кількості елементів масивів даних»

Тема: «Основні алгоритми для обчислення суми та підрахунку кількості елементів масивів даних».

Мета: Визначити ключові правила та принципи створення алгоритмів для обробки елементів масивів даних за допомогою графічного подання та сформувати практичні навички роботи з редактором блок-схем алгоритмів.

Обладнання: комп'ютер типу Intel Core i3-i5 з ОС Windows (10, 11), редактор блок-схем алгоритмів.

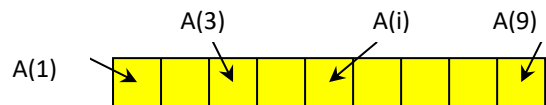
Теоретичні основи:

Основні характеристики масиву

Масив характеризується ім'ям, розмірністю і розміром.

Ім'я масиву

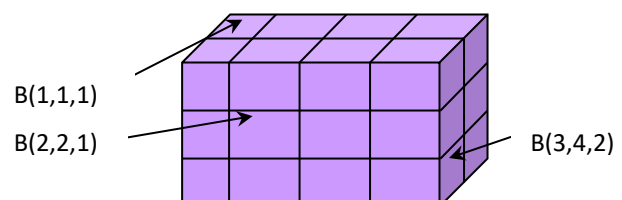
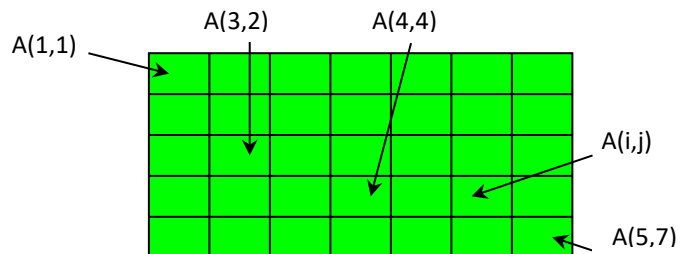
- Ім'я масиву формується за загальними правилами для іменування змінних і виступає як ідентифікатор.
- Воно не повинно збігатися з іменами інших змінних, що використовуються у тій же програмі.
- Елементи масиву позначаються через ім'я масиву з індексом у круглих дужках, наприклад: $A(1)$, $B(8)$, $C(3,5)$.
- В якості індексів можуть використовуватися константи ($A(1)$, $B(8)$, $C(3,5)$), змінні ($F(i)$, $MAS(i,j)$) або індексні арифметичні вирази ($MAS(i+4)$).
- Якщо використовуються змінні або індексні вирази, значення всіх цих змінних повинні бути визначені до звернення до елементів масиву.
- В одному масиві всі елементи повинні бути одного типу (наприклад, $A(14)$, $B(8)$, $C(3,5)$); базовий тип елемента визначає тип всього масиву.



Розмірність масиву

Індекси вказують на розташування елементів у масиві, а число індексів в імені елемента задає розмірність масиву, тобто спосіб організації його структури:

- Одновимірний масив — має лінійну структуру, його елемент позначається змінною з одним індексом, наприклад: $A(1)$, $A(i)$.
- Двовимірний масив — організований у вигляді прямокутної таблиці, його елемент позначається змінною з двома індексами, наприклад: $A(3,4)$, $A(i,j)$, де перший індекс



відповідає номеру рядка, а другий — номеру стовпця таблиці.

- Тривимірний масив можна уявити як елементи, розташовані у кубічній структурі.

Індекс — це перелік числових виразів, розділених комами.

Розмір масиву

Щоб зберегти елементи масиву в пам'яті комп'ютера, потрібно виділити відповідну кількість комірок пам'яті, яка визначається розміром масиву.

Розмір масиву визначається верхніми межами індексів у кожному вимірі:

- Для одномірного масиву — наприклад, якщо останній елемент масиву $A(19)$, його розмір дорівнює 19.
- Для двовимірного або багатовимірного масиву — розмір обчислюється як добуток верхніх меж індексів. Наприклад, якщо останній елемент масиву $A(5,7)$, його розмір дорівнює $5 \cdot 7 = 35$; для масиву $B(3,4,2)$ — $3 \cdot 4 \cdot 2 = 24$.

Доступ до елементів масиву

- Ми звертаємося до окремих елементів масиву, щоб присвоїти їм значення, вивести на друк або використовувати у потрібних обчисленнях, так само, як ми працюємо з окремими змінними.
- Щоб звернутися до конкретного елемента масиву, необхідно вказати ім'я масиву та в дужках (круглих або квадратних) номер цього елемента. Значення в дужках називають індексом масиву. Максимальний індекс відповідає значенню, заданому при оголошенні масиву.
- Наприклад, перший елемент масиву A позначається як $A(1)$, другий — $A(2)$, а 53-й — $A(53)$.
- Потужним засобом при роботі з масивами є можливість використовувати індекс як змінну або складний вираз, наприклад: $A(i,j)$, $MAS(i+1)$. При цьому слід забезпечити, щоб значення індексу завжди відповідало розміру масиву.
- Програма не повинна звертатися до елементів із індексами менше 0 або більшими за розмір масиву, оскільки це є помилкою і називається виходом

за межі масиву.

Завантаження багатомірних масивів:

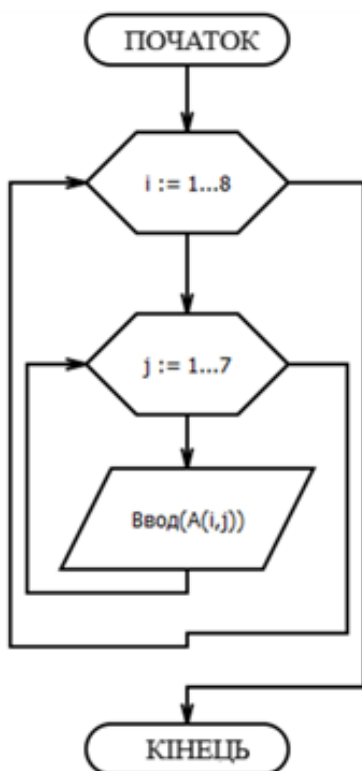
Значення елементів багатовимірних масивів можна задавати тими ж способами, що й для звичайних змінних. До них належать:

- команда присвоєння ($x := 1$),
- команда введення значень (ввод x).

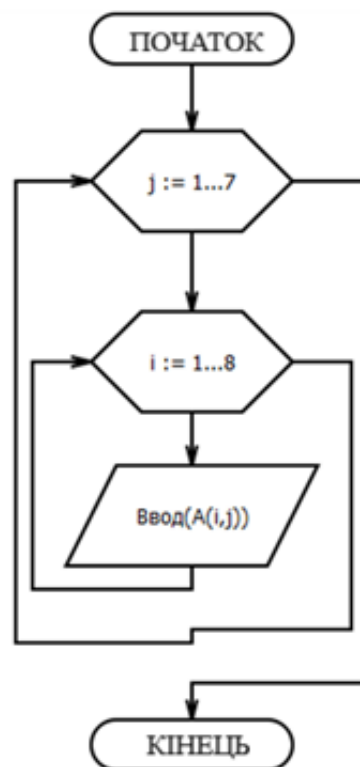
Найзручніший спосіб присвоєння значень елементам масивів — використання цих команд всередині циклу. Для двовимірних масивів зазвичай застосовують два вкладені цикли, найчастіше з параметрами: лічильник одного циклу перебирає рядки масиву, а лічильник іншого — стовпці.

Нижче наведено приклади присвоєння значень елементам двовимірного масиву, наприклад $A(8,7)$ (загальний вигляд — $A(i,j)$):

Блок-схема алгоритму введення даних по рядках (схема А)



Блок-схема алгоритму введення даних по стовпцях (схема Б)



У псевдокодi алгоритми представлені у такій формі:

алг Алгоритм А	алг Алгоритм Б
нач	нач
нц для і от 1 до 8	нц для j от 1 до 7
нц для j от 1 до 7	нц для і от 1 до 8
Ввод(A(i,j))	Ввод(A(i,j))
кц	кц
кц	кц
кон	кон

Приклади виконання завдання:

Задача №1: Побудувати блок-схему алгоритму для обчислення суми елементів одномірного масиву A(17)

Пояснення до розробки алгоритму розв’язування задачі:

Перед виконанням будь-яких операцій з елементами масиву необхідно спочатку ввести їхні значення. Введення елементів, так само як і підрахунок суми, має циклічний характер. Тому операції введення чергового елемента A(i) та додавання його до змінної суми S входять до тіла одного циклу.

Чому підрахунок суми розміщують у циклі? Розглянемо схему підрахунку: спочатку змінна суми S не повинна мати жодного значення. Якщо раніше вона використовувалася в програмі, щоб уникнути випадкового значення, її потрібно обнулити: $S = 0$.

У циклі накопичення значень відбувається наступне: до попереднього значення S додається поточне значення елемента A(i).

Схематично це виглядає в такий спосіб:

- Перший прохід циклу: $S = [0] + A(1)$, де [0] — початкове значення змінної S.
- Другий прохід циклу: $S = [0 + A(1)] + A(2)$, де $[0 + A(1)]$ — попереднє значення S після першого проходу.

- Третій прохід циклу: $S = [0 + A(1) + A(2)] + A(3)$, де $[0 + A(1) + A(2)]$ — попереднє значення S після другого проходу.
- Четвертий прохід циклу: $S = [0 + A(1) + A(2) + A(3)] + A(4)$, де $[0 + A(1) + A(2) + A(3)]$ — попереднє значення S після третього проходу.

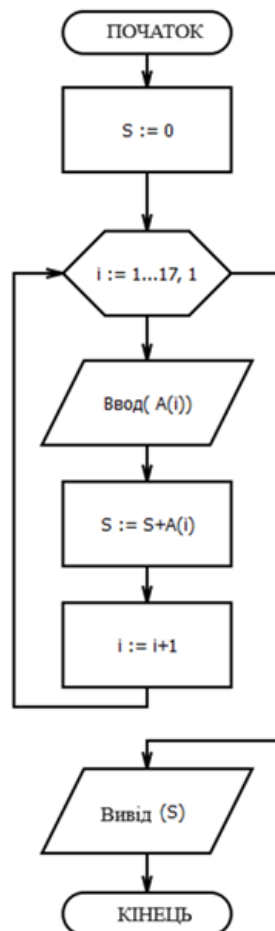
І так далі. Узагальнюючи процес накопичення суми, його можна записати у вигляді формули:

$$S = S + A(i)$$

Висновок. Значення змінної суми обчислюється всередині циклу, щоб отримати кінцевий результат S . Якщо ж команду виведення значення S розмістити всередині циклу, на екран будуть виводитися всі проміжні значення змінної після кожного проходу циклу.

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок- Псевдокод, поданий алгоритмічною мовою:



алг Сума_елементів

нач

| S := 0

| нц для i от 1 до 17

| | ввод A(i)

| | S:=S+A(i)

| | i:=i+1

| кц

| вивід S

кон

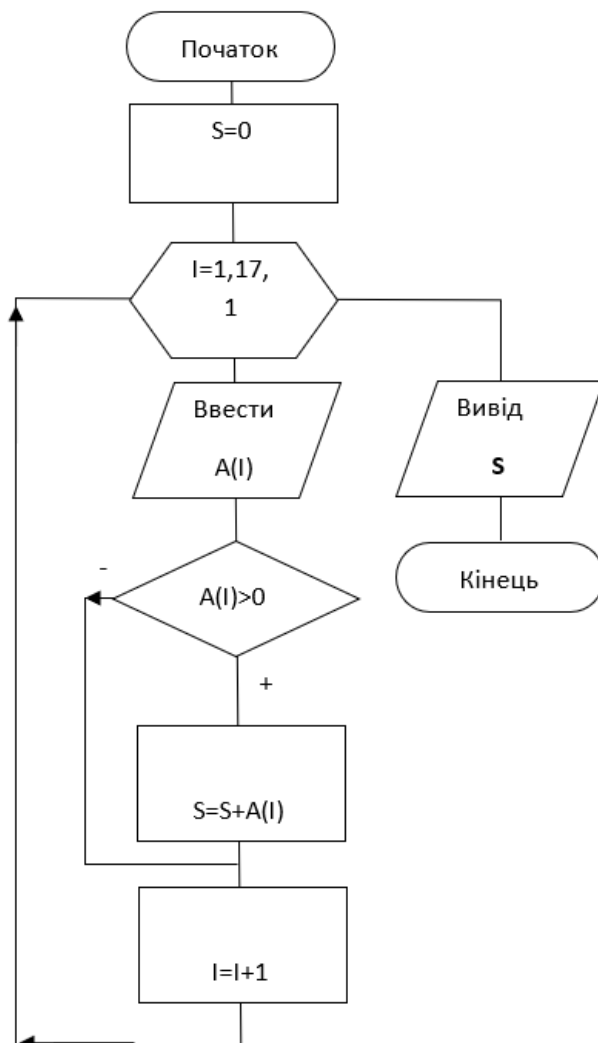
Задача №2: Побудувати блок-схему алгоритму для обчислення суми елементів одномірного масиву $A(17)$ з урахуванням умови.

Пояснення до розробки алгоритму розв'язування задачі:

Припустимо, потрібно обчислити суму лише додатних елементів масиву A . Для цього в тіло циклу вставляємо команду розгалуження (умовного переходу), де задаємо умову, за якою буде здійснюватися підрахунок, наприклад, враховуються лише додатні елементи. У даному випадку використовується команда неповного умовного переходу: якщо умова виявляється хибною (поточний елемент від'ємний), дія гілки не виконується, і цей елемент просто пропускається.

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок-схема: Псевдокод, поданий алгоритмічною мовою:



```

алг
Сума_позитивних_чисел
нач
| S := 0
| нц для i от 1 до 17
| | Ввести A(i)
| | если A(i) > 0
| | | то
| | | S := S+A(i)
| | | иначе
| | все
| | i := i+1
| кц
| Вивод S
кон
  
```

Задача №3: Побудувати блок-схему алгоритму для підрахунку кількості елементів одномірного масиву $A(17)$.

Пояснення до розробки алгоритму розв’язування задачі:

Принцип підрахунку кількості елементів у масиві A схожий на підрахунок суми елементів: при кожному проході циклу до попереднього значення змінної N додається одиниця (1), фактично «підраховуючи» елементи. Як і змінну S під час підрахунку суми, змінну N перед початком циклу слід обнулити, щоб уникнути спотворення результату.

Схематично алгоритм підрахунку кількості можна зобразити наступним чином:

$$N = 0$$

У циклі підрахунок значень змінної N (фактично визначення поточного значення кількості елементів) відбувається за таким алгоритмом — до попереднього значення N додається одиниця. Схематично це виглядає так:

- Перший прохід циклу:

$$N = [0] + 1, \text{ де } [0] \text{ — початкове значення } N \text{ перед початком циклу.}$$

- Другий прохід циклу:

$$N = [0 + 1] + 1, \text{ де } [0 + 1] \text{ — попереднє значення } N \text{ після першого проходу.}$$

- Третій прохід циклу:

$$N = [0 + 1 + 1] + 1, \text{ де } [0 + 1 + 1] \text{ — попереднє значення } N \text{ після другого проходу.}$$

- Четвертий прохід циклу:

$$N = [0 + 1 + 1 + 1] + 1, \text{ де } [0 + 1 + 1 + 1] \text{ — попереднє значення } N \text{ після третього проходу.}$$

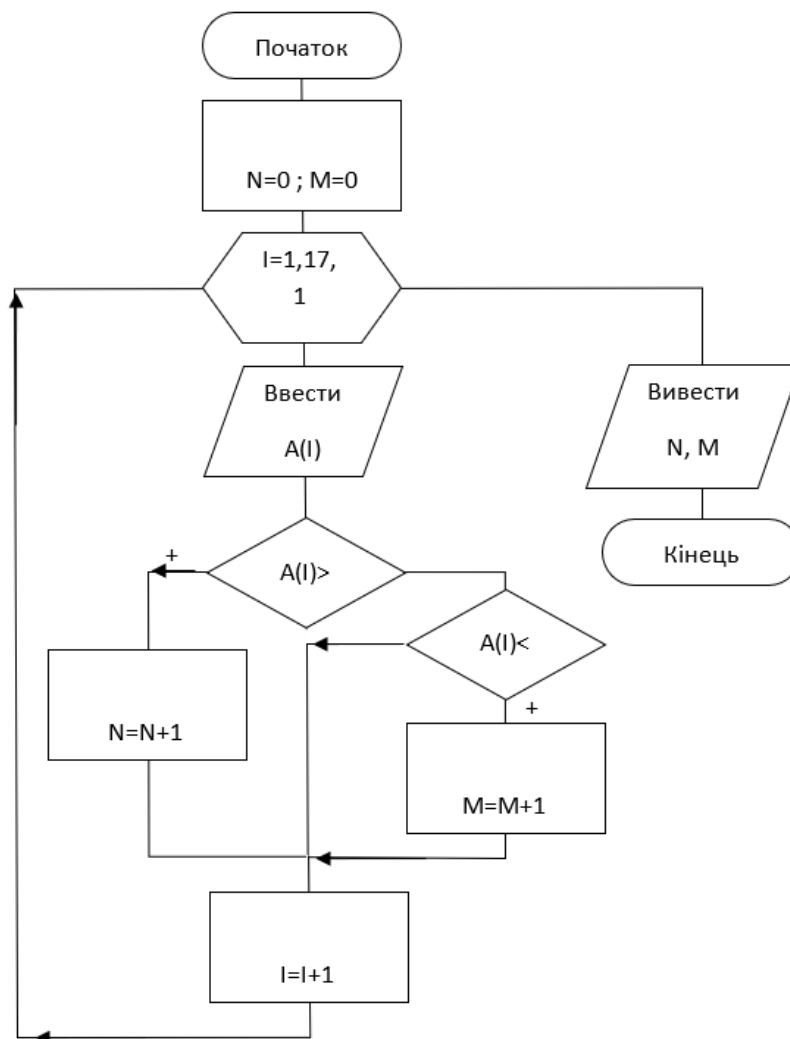
І так далі. Узагальнюючи процес підрахунку кількості елементів, його можна записати формулою:

$$N = N + 1$$

Якщо потрібно підраховувати елементи за певною умовою, використовують команду умовного переходу (повний або неповний), де задають відповідну умову підрахунку, як було описано раніше.

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок- Псевдокод, поданий алгоритмічною мовою:



```

алг Алгоритм
визначення кількості
нач
| N := 0
| M := 0
| нц для i от 1 до 17
| | ввести A(i)
| | якщо A(i) > 0
| | | то
| | | N := N+1
| | | інакше
| | | якщо A(i) < 0
| | | | то
| | | | M := M+1
| | | | інакше
| | | все
| | все
| | i := i+1
| кц
| вивести N, M
кон
  
```

Порядок виконання роботи:

1. Скласти алгоритм розв'язання задачі словесним способом та оформити його у звіті;
2. Створити блок-схему за допомогою редактора блок-схем алгоритмів;
3. Внести команди у відповідні елементи блок-схеми;
4. Відобразити блок-схему у звіті графічно;
5. Описати алгоритм розв'язання задачі у вигляді псевдокоду;
6. Продемонструвати готове рішення задачі викладачу в редакторі блок-схем алгоритмів.

Завдання №1: Побудувати блок-схему алгоритму для підрахунку кількості додатних і від'ємних елементів в одномірному числовому масиві $A(15)$:

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок- Псевдокод, поданий алгоритмічною
схема: мовою:

Відмітка про виконання завдання:

Завдання №2: Побудувати блок-схему алгоритму для обчислення суми елементів рядків двовимірного масиву $MAC(9,5)$:

Алгоритм розв'язання задачі представлено у вигляді словесного опису:

Процес розв'язання задачі у вигляді алгоритму:

Графічна форма подання — блок- Псевдокод, поданий алгоритмічною
схема: мовою:

Відмітка про виконання завдання:

Контрольні питання:

1. Як визначити поняття «масив»?
2. Які існують типи масивів?
3. Назвіть основні характеристики масивів.
4. Що розуміють під «ім'ям масиву»?
5. Як визначається «розмірність масиву»?
6. Що таке «розмір масиву»?
7. Для чого використовують команди розгалуження (умовного переходу) у циклі при роботі з елементами масиву?
8. Які блок-схемні конструкції відображають операції введення/виведення елементів одновимірного масиву? Як це записується у псевдокоді?
9. Які блок-схемні конструкції описують дії з елементами масиву за певною умовою? Як це оформлюється у псевдокоді?
10. Які конструкції блок-схеми використовують для введення/виведення елементів двовимірного масиву? Як це виглядає у псевдокоді?
11. Як виконуються операції над рядками та стовпцями двовимірних масивів?
12. Що таке «вкладені цикли» в алгоритмі?

Висновок:

Висновки до розділу

В даному розділі наведено розробки п'яти практичних робіт за темами: «Інструменти побудови алгоритму та принципи розробки блок-схем», «Базова структура алгоритму – лінійна», «Базова структура алгоритму – розгалуження», «Базова структура алгоритму – циклічна», «Основні алгоритми для обчислення суми та підрахунку кількості елементів масивів даних» з використанням розробленого у другому розділі редактора блок-схем алгоритмів.

Узагальнюючи результати дослідження дає підстави сформулювати такі практичні рекомендації:

- у процесі навчання алгоритмізації доцільно передбачати в межах кожної теми етап конструювання блок-схеми з подальшим її колективним обговоренням та рефлексивним аналізом;
- забезпечувати послідовний перехід від найпростіших схем, що відображають лінійні алгоритми, до більш складних структур, зокрема розгалужень, вкладених циклів та ітераційних процедур;
- застосовувати блок-схеми як засіб педагогічного моделювання навчальних занять, зокрема під час пояснення нового матеріалу або аналізу типових помилок учнів;
- поєднувати опанування алгоритмів із проєктно-орієнтованою діяльністю, у межах якої учні розробляють дидактичні матеріали на основі створених блок-схем.

Отже, використання блок-схем у процесі навчання інформатики слід розглядати не як допоміжний елемент, а як інтегрований складник освітнього процесу, що сприяє розвитку алгоритмічного мислення, формуванню методичної рефлексії та підвищенню рівня цифрової компетентності учнів.

ВИСНОВКИ

У магістерській роботі було розглянуто використання редактора блок-схем алгоритмів у навчанні основ алгоритмізації в закладах загальної середньої освіти.

Проведене дослідження дало змогу визначити комплекс педагогічних умов, за яких навчання побудови блок-схем набуває ефективності як засіб формування алгоритмічного мислення в здобувачів освіти. Узагальнення результатів аналізу наукових джерел, спостережень за організацією освітнього процесу дозволило виокремити три взаємопов'язані групи умов: когнітивно-методичні, організаційно-технологічні та професійно спрямовані.

До когнітивно-методичних умов віднесено забезпечення поетапного переходу від словесного опису задачі до її формалізованого подання шляхом побудови блок-схеми, яка слугує засобом упорядкування мислення. На цьому етапі особливої ваги набуває чітке дотримання правил створення блок-схем, знання типів блоків, умовних позначень і зв'язків між ними, що запобігає формальному підходу та сприяє глибшому розумінню логіки алгоритму. У такому контексті блок-схема виступає посередником між мовою задачі та мовою програмування, даючи змогу зосередитися на структурі розв'язання до переходу безпосередньо до кодування.

Організаційно-технологічні умови передбачають використання доступних і зручних програмних засобів для створення блок-схем, інтерфейс яких не відволікає учнів від осмислення алгоритмічної логіки. Практична діяльність має поєднувати індивідуальну та групову роботу з побудови блок-схем, супроводжувану рефлексією, взаємооцінюванням і презентацією результатів. Такий підхід сприяє розвитку критичного мислення, аналітичних умінь і комунікативних навичок. Особливо результативним є застосування проєктних методів, за яких учні спочатку моделюють алгоритми у вигляді блок-схем, а згодом реалізують їх у програмному коді.

Професійно орієнтовані умови пов'язані з включенням елементів методичної підготовки, зокрема аналізу типових помилок учнів, обговорення

доцільності використання блок-схем на різних етапах навчання програмуванню, а також моделювання окремих фрагментів уроків. Таке спрямування активізує педагогічну рефлексію викладачів і формує готовність адаптувати візуальні методи навчання відповідно до вікових та пізнавальних особливостей здобувачів освіти.

Отже, результативність застосування блок-схем у процесі розвитку алгоритмічного мислення забезпечується дотриманням взаємопов'язаної тріади: усвідомленості навчальних дій (когнітивно-методичний рівень), наявності технологічної підтримки діяльності (організаційний рівень) і професійного моделювання майбутньої педагогічної практики (методико-рефлексивний рівень). У сукупності ці умови формують освітнє середовище, яке сприяє не лише засвоєнню алгоритмічних структур, а й розвитку здатності працювати з ними на рівні розуміння, пояснення та подальшого навчання інших.

Розроблено та реалізовано редактор блок-схем алгоритмів. Розроблений прототип інтерактивної побудови блок-схем дозволяє шляхом маніпулювання графічними об'єктами редагувати логіку програм, автоматично формувати програмні коди для різних мов програмування (Pascal, C / C ++, Алгоритмічна мова, PHP, JavaScript, Python). Програма написана на мові C ++ на основі бібліотеки Qt 4.

Розроблений програмний продукт редактор блок-схем алгоритмів включає в себе вирішення певних задач, зокрема:

- 1) аналіз зв'язку блок-схем із програмним кодом;
- 2) створення і редагування блок-схем;
- 3) професійно розроблені шаблони для прискорення робочого процесу;
- 4) проста візуалізація даних без складного програмного забезпечення, яке треба вивчати;
- 5) генерування вихідного коду на основі блок-схеми алгоритму.

Узагальнюючи отримані результати, можна запропонувати такі практичні рекомендації:

- під час навчання алгоритмізації включати до кожної теми етап побудови

блок-схеми із наступним її обговоренням і рефлексією;

- забезпечити поетапний перехід від простих схем (лінійні алгоритми) до складніших (розгалуження, вкладені цикли, ітераційні процедури);
- використовувати блок-схеми як інструмент педагогічного моделювання уроків, зокрема при поясненні нових тем або аналізі типових учнівських помилок;
- поєднувати вивчення алгоритмів із проектною діяльністю, де учні створюють дидактичні матеріали на основі побудованих блок-схем.

Таким чином, побудова блок-схем у процесі підготовки учнів на уроках інформатики має розглядатися не як допоміжний, а як інтегрований компонент навчання, що забезпечує розвиток алгоритмічного мислення, методичної рефлексії та цифрової компетентності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Coşkunserçe, O. Comparing the use of block-based and robot programming in introductory programming education: Effects on perceptions of programming self-efficacy. *Computer Applications in Engineering Education*, 31(5), 1234–1255.
2. Espinal, A., Vieira, C., & Guerrero-Bequis, V. Student ability and difficulties with transfer from a block-based programming language into other programming languages: A case study in Colombia. *Computer Science Education*, 33(4), 567–599.
3. Khvorostina, Y., Shamonina, V., & Semenikhina, O. The connection between the study of mathematics and programming through the prism of scientific and pedagogical research. *Вісник науки та освіти*, 4(34), 932–945.
4. Rudenko, Y., Drushlyak, M., Osmuk, N., Shvets, O., Kolyshkin, O., & Semenikhina, O. Problems of teaching pupils of non-specialized classes to program and ways to overcome them: Local study. *International Journal of Computer Science and Network Security*, 22(1), 105–112.
5. Sanusi, I. T., Cudjoe, E. S., Ayanwale, M. A., & Adepoju, B. Pre-service teachers' perception of programming education. *SAGE Open*, 15(1).
6. Semenikhina, O. V., & Rudenko, Y. O. Проблеми навчання програмувати учнів старших класів та шляхи їх подолання. *Інформаційні технології і засоби навчання*, 66(4), 54–64.
7. Umezawa, K., Ishida, K., Nakazawa, M., & Hirasawa, S. Proposal and evaluation of intermediate content for the transition from visual to text-based programming languages. In T. X. Bui (Ed.), *Proceedings of the 56th Hawaii International Conference on System Sciences* (pp. 83–92).
8. Кобильник, Т., Когут, У., & Жидик, В. Методичні аспекти вивчення основ алгоритмізації і програмування мовою Python у шкільному курсі інформатики у старших класах. *Фізико-математична освіта*, 31(5), 36–44.
9. Дегтярьова, Н., Петренко, С., & Удовиченко, О. Робота з графічними віджетами при вивченні мови програмування Python в закладах загальної

- середньої освіти. Освіта. Інноватика. Практика, 11(4), 26–34.
10. Пенко, В., & Пенко, О. Використання візуалізації на різних етапах вивчення дисципліни «Програмування». Освіта. Інноватика. Практика, 11(2), 31
 11. Кобильник, Т., Когут, У., & Жидик, В. Методичні аспекти вивчення основ алгоритмізації і програмування мовою Python у шкільному курсі інформатики у старших класах. Фізико-математична освіта, 31(5), 36–44.
 12. Юрченко, А.О., Беспалий, В.Р., & Семеніхіна, О.В. Навчання програмуванню в цифрову епоху: роль МООС і візуалізації у формуванні алгоритмічної грамотності. Суспільство та національні інтереси, 8(16), 325-335.
 13. Юрченко, А. О., Семеніхіна, О. В., Хворостіна, Ю. В., & Удовиченко, О. М. Навчання програмувати в старшій школі крізь призму чинних навчальних програм. Фізико-математична освіта, 2(20), Ч.2, 47–54.
 14. Базурін В. М. Середовища програмування як засіб навчання учнів основ програмування. Інформаційні технології і засоби навчання. 2017. Т. 59, вип. 3. С. 13–27.
 15. Жалдак М. І. Інформатика – фундаментальна наукова дисципліна. Вона має вивчати закони природи, інформаційні процеси і відповідні технології (продовження). Комп'ютер у школі та сім'ї. 2010, № 3. С. 7–11.
 16. Жалдак М. І. Система підготовки вчителя до використання інформаційно-комунікаційних технологій в навчальному процесі. Науковий часопис НПУ імені М. П. Драгоманова. Серія 2 : Комп'ютерно-орієнтовані системи навчання. 2011, №. 11. С. 3–15.
 17. Семеніхіна О.В., Руденко Ю.О. Проблеми навчання програмувати учнів старших класів та шляхи їх подолання. Інформаційні технології і засоби навчання. 2018. Том 66, №4. С.54–64.
 18. Яку мову програмування вивчати у школі. Комп'ютер у школі та сім'ї. 2013. №7. С.14–18.

- 19.J. Monsálvez. "Python como primer lenguaje de programación textual en la enseñanza secundaria", *Educationn the Knowledge Society*, vol 18, n. 2, pp.147-162. 2017.
- 20.B. Fagan, and B. Payne, "Learning to Program in Python – by Teaching It!", in *Proceedings of the Interdisciplinary STEM Teaching and Learning Conference*, Vol. 1, pp.99-107, 2017.
- 21.Р. Н. Shevchuk, «Основні підходи добору мови та середовища програмування як засобів навчання», *ITLT*, вип. 17, вип. 3, Вер 2010. doi:
- 22.V. М. Bazurin, «Середовища програмування як засіб навчання учнів основ програмування», *ITLT*, вип. 59, вип. 3, с. 13-27, Чер 2017. doi:
- 23.О. V. Semenykhina, та Rudenko Y. O., «Проблеми навчання програмувати учнів старших класів та шляхи їх подолання», *ITLT*, вип. 66, вип. 4, с. 54-64, Вер 2018.
- 24.О. Маловічко, та С. Конюхов, "Застосування спеціалізованого педагогічного програмного комплексу у процесі вивчення програмування у восьмому класі", *Ukrainian Journal of Educational Studies and Information Technology*, 5 (4), с. 38-55, 2017.
- 25.Л. Міцкан, Т. Вербицька, та В. Базурін, "Порівняльний аналіз мов Python і Free Pascal як перших мов програмування для учнів 8 класу", *Актуальні питання природничо-математичної освіти*, № 2 (10), с. 130-139, 2017.

ДОДАТОК А

Лістинг кодуmainwindow.cpp

```

#include "mainwindow.h"
#include "zvflowchart.h"
#include <QtGui>
#include <QtSvg>
#include <QSettings>
#include "qflowchartstyle.h"
#include "sourcecodegenerator.h"
#include <QWidgetList>
#include <QtPrintSupport/QtPrinter>
#include <QtPrintSupport/QtPrintDialog>
QString afceVersion()
{
    return PROGRAM_VERSION;
}
void AfcScrollArea::mousePressEvent(QMouseEvent *event)
{
    event->accept();
    emit mouseDown();
}
void AfcScrollArea::wheelEvent(QWheelEvent *event)
{
    if((event->modifiers() & Qt::ControlModifier) != 0) {
        event->ignore();
        emit zoomStepped(event->delta() / 120);
    }
    else {
        if((event->modifiers() & Qt::ShiftModifier) == 0) {
            QScrollBar * vsb = verticalScrollBar();
            if (vsb != NULL) {
                /* scroll vertically */
                vsb->setValue(vsb->value() - event->delta());
            }
        }
        else {
            QScrollBar * gsb = horizontalScrollBar();
            if (gsb != NULL) {
                /* scroll horizontally */
                gsb->setValue(gsb->value() - event->delta());
            }
        }
        event->accept();
    }
}
MainWindow::MainWindow(QWidget *parent, Qt::WindowFlags flags)
: QMainWindow(parent, flags), fDocument(0)
{
    #if defined(Q_WS_X11) or defined(Q_OS_LINUX)
        QDir::setSearchPaths("generators", QStringList() << QString(PROGRAM_DATA_DIR)
+ "generators");
    #endif
}

```



```

#else
    QDir::setSearchPaths("generators", QStringList() << qApp->applicationDirPath() +
"/generators");
#endif
    setupUi();
    readSettings();
    retranslateUi();
    QFlowChart *fc = new QFlowChart(this);
    setDocument(fc);
    document()->setZoom(1);
    connect(document(), SIGNAL(statusChanged()), this, SLOT(slotStatusChanged()));
    connect(document(), SIGNAL(editBlock(QBlock *)), this, SLOT(slotEditBlock(QBlock
*)));
    connect(actUndo, SIGNAL(triggered()), document(), SLOT(undo()));
    connect(actRedo, SIGNAL(triggered()), document(), SLOT(redo()));
    connect(document(), SIGNAL(changed()), this, SLOT(updateActions()));
    connect(document(), SIGNAL(changed()), this, SLOT(generateCode()));
    document()->setStatus(QFlowChart::Selectable);
    connect(saScheme, SIGNAL(mouseDown()), document(), SLOT(deselectAll()));
    connect(codeLanguage, SIGNAL(activated(int)), this, SLOT(codeLangChanged(int)));
    QFlowChartStyle st;
    QPalette pal = palette();
    st.setLineWidth(2);
    st.setNormalBackground(pal.color(QPalette::Base));
    st.setNormalForeground(pal.color(QPalette::WindowText));
    st.setNormalMarker(Qt::red);
    st.setSelectedBackground(pal.color(QPalette::Highlight));
    st.setSelectedForeground(pal.color(QPalette::HighlightedText));
    st.setNormalMarker(Qt::green);
    st.setFontSize(10);
    document()->setChartStyle(st);
    labelMenu = new QLabel(statusBar());
    statusBar()->setSizeGripEnabled(false);
    statusBar()->addWidget(labelMenu);
    labelMenu->setAlignment(Qt::AlignCenter);
    isSaved = true; //Let's allow to close application if no modification were made in empty
document
    connect(document(), SIGNAL(modified()), SLOT(slotDocumentChanged()));
    connect(this, SIGNAL(documentLoaded()), SLOT(slotDocumentLoaded()));
    connect(this, SIGNAL(documentSaved()), SLOT(slotDocumentSaved()));
    if (qApp->arguments().size() > 1) {
        QFile test(qApp->arguments().at(1));
        if(test.exists()) {
            slotOpenDocument(qApp->arguments().at(1));
        }
        else {
            QMessageBox::critical(this, tr("Failed to open a file"), tr("Unable to open file
'%1'.").arg(qApp->arguments().at(1)));
        }
    }
}
void MainWindow::writeSettings()
{

```

```

    QSettings settings("afce", "application");
    settings.setValue("geometry", geometry());
    settings.setValue("windowState", saveState());
}
void MainWindow::closeEvent(QCloseEvent *event)
{
    if (okToContinue()) {
        writeSettings();
        event->accept();
    } else {
        event->ignore();
    }
}
void MainWindow::readSettings()
{
    QSettings settings("afce", "application");

    setGeometry(settings.value("geometry", QRect(100, 100, 800, 600)).toRect());
    restoreState(settings.value("windowState").toByteArray());
}
void MainWindow::setupUi()
{
    QApplication::setWindowIcon(QIcon(":/images/icon.png"));
    createActions();
    createMenu();
    createToolBar();
    QWidget *body = new QWidget;
    QWidget *zoomPanel = new QWidget;
    zoomPanel->setMinimumHeight(18);
    saScheme = new AfcScrollArea();
    QPalette pal = saScheme->palette();
    pal.setColor(QPalette::Window, pal.color(QPalette::Base));
    saScheme->setPalette(pal);
    setCentralWidget(body);
    QVBoxLayout *bodyLayout = new QVBoxLayout;
    bodyLayout->addWidget(saScheme);
    bodyLayout->addWidget(zoomPanel);
    body->setLayout(bodyLayout);
    zoomSlider = new QSlider(Qt::Horizontal, zoomPanel);
    zoomLabel = new QLabel;
    QHBoxLayout *zoomLayout= new QHBoxLayout;
    zoomLayout->addStretch();
    zoomLayout->addWidget(zoomLabel);
    zoomLayout->addWidget(zoomSlider);
    zoomPanel->setLayout(zoomLayout);
    zoomSlider->setRange(1, 20);
    zoomSlider->setSingleStep(1);
    zoomSlider->setPageStep(10);
    connect(zoomSlider, SIGNAL(valueChanged(int)),this, SLOT(setZoom(int)));
    zoomSlider->setValue(4);
    connect(saScheme, SIGNAL(zoomStepped(int)), this, SLOT(shiftZoom(int)));
    connect(saScheme, SIGNAL(scrollStepped(int)), this, SLOT(shiftScrollY(int)));
}

```

```

createToolbox();
dockCode = new QDockWidget(this);
dockCode->setObjectName("dock_code");
dockCode->setAllowedAreas(Qt::AllDockWidgetAreas);
addDockWidget(Qt::RightDockWidgetArea, dockCode);
codeWidget = new QFrame;
codeLanguage = new QComboBox;
codeText = new QTextEdit;
codeLabel = new QLabel;
actCode->setCheckable(true);
actCode->setChecked(dockCode->isVisible());
connect(actCode, SIGNAL(triggered(bool)), dockCode, SLOT(setVisible(bool)));
connect(dockCode, SIGNAL(visibilityChanged(bool)), actCode,
SLOT(setChecked(bool)));
//connect(dockCode, SIGNAL(visibilityChanged(bool)), this,
SLOT(docCodeVisibilityChanged(bool)));
codeWidget->setFrameStyle(QFrame::StyledPanel | QFrame::Plain);
codeLabel->setBuddy(codeLanguage);
codeText->setFont(QFont("Courier New", 12));
codeText->setLineWrapMode(QTextEdit::NoWrap);
codeText->setReadOnly(true);
QVBoxLayout * vbl = new QVBoxLayout;
vbl->addWidget(codeLabel);
vbl->addWidget(codeLanguage);
vbl->addWidget(codeText);
codeWidget->setLayout(vbl);
dockCode->setWidget(codeWidget);
helpWindow = new THelpWindow();
helpWindow->setObjectName("help_window");
helpWindow->setAllowedAreas(Qt::AllDockWidgetAreas);
addDockWidget(Qt::RightDockWidgetArea, helpWindow);
helpWindow->hide();
actHelp->setCheckable(true);
actHelp->setChecked(helpWindow->isVisible());
connect(actHelp, SIGNAL(triggered(bool)), helpWindow, SLOT(setVisible(bool)));
connect(helpWindow, SIGNAL(visibilityChanged(bool)), actHelp,
SLOT(setChecked(bool)));
}
QToolButton * createToolButton(const QString & fileName)
{
    QToolButton *Result = new QToolButton;
    Result->setIconSize(QSize(32, 32));
    Result->setIcon(QIcon(fileName));
    Result->setToolButtonStyle(Qt::ToolButtonTextBesideIcon);
    Result->setAutoRaise(true);
    Result->setSizePolicy(QSizePolicy::Minimum, QSizePolicy::Fixed);
    Result->setCheckable(true);
    Result->setAutoExclusive(true);
    return Result;
}
void MainWindow::createToolbox()
{
    dockTools = new QDockWidget(this);

```

```

dockTools->setObjectName("dock_tools");
dockTools->setAllowedAreas(Qt::LeftDockWidgetArea | Qt::RightDockWidgetArea);
dockTools->setMinimumWidth(150);
addDockWidget(Qt::LeftDockWidgetArea, dockTools);
connect(dockTools, SIGNAL(visibilityChanged(bool)), this,
SLOT(docToolsVisibilityChanged(bool)));
    tbArrow = createToolButton(":/images/arrow.png");
    tbArrow->setChecked(true);
    tbProcess = createToolButton(":/images/simple.png");
    tbAssign = createToolButton(":/images/assign.png");
    tbIf = createToolButton(":/images/if.png");
    tbFor = createToolButton(":/images/for.png");
    tbWhilePre = createToolButton(":/images/while.png");
    tbWhilePost = createToolButton(":/images/until.png");
    tbIo = createToolButton(":/images/io.png");
    tbOu = createToolButton(":/images/ou.png");
    tbForCStyle = createToolButton(":/images/forc.png");
    connect(tbArrow, SIGNAL(pressed()), this, SLOT(slotToolArrow()));
    connect(tbProcess, SIGNAL(pressed()), this, SLOT(slotToolProcess()));
    connect(tbAssign, SIGNAL(pressed()), this, SLOT(slotToolAssign()));
    connect(tbIf, SIGNAL(pressed()), this, SLOT(slotToolIf()));
    connect(tbFor, SIGNAL(pressed()), this, SLOT(slotToolFor()));
    connect(tbWhilePre, SIGNAL(pressed()), this, SLOT(slotToolWhilePre()));
    connect(tbWhilePost, SIGNAL(pressed()), this, SLOT(slotToolWhilePost()));
    connect(tbIo, SIGNAL(pressed()), this, SLOT(slotToolIo()));
    connect(tbOu, SIGNAL(pressed()), this, SLOT(slotToolOu()));
    connect(tbForCStyle, SIGNAL(pressed()), this, SLOT(slotToolForCStyle()));
    toolsWidget = new QFrame;
    toolsWidget->setFrameStyle(QFrame::StyledPanel | QFrame::Plain);
    QVBoxLayout *tl = new QVBoxLayout;
    tl->setSpacing(2);
    tl->addWidget(tbArrow);
    tl->addWidget(tbIo);
    tl->addWidget(tbOu);
    tl->addWidget(tbProcess);
    tl->addWidget(tbAssign);
    tl->addWidget(tbIf);
    tl->addWidget(tbFor);
    tl->addWidget(tbWhilePre);
    tl->addWidget(tbWhilePost);
    tl->addWidget(tbForCStyle);
    tl->addStretch();
    toolsWidget->setLayout(tl);
    dockTools->setWidget(toolsWidget);
    actTools->setCheckable(true);
    actTools->setChecked(dockTools->isVisible());
    connect(actTools, SIGNAL(triggered(bool)), dockTools, SLOT(setVisible(bool)));
    connect(dockTools, SIGNAL(visibilityChanged(bool)), actTools,
SLOT(setChecked(bool)));
}
void MainWindow::retranslateUi()
{
    dockTools->setWindowTitle(tr("Tools"));
}

```

```

tbArrow->setText(tr("Select"));
tbProcess->setText(tr("Process"));
tbAssign->setText(tr("Assign"));
tbIf->setText(tr("If...then...else"));
tbFor->setText(tr("FOR loop"));
tbWhilePre->setText(tr("loop with pre-condition"));
tbWhilePost->setText(tr("loop with post-condition"));
tbIo->setText(tr("Input"));
tbOu->setText(tr("Output"));
tbForCStyle->setText(tr("FOR loop (C/C++)"));
actExit->setText(tr("E&xit"));
actExit->setStatusTip(tr("Exit from program"));
actOpen->setText(tr("&Open..."));
actOpen->setStatusTip(tr("Open saved file"));
actSave->setText(tr("&Save"));
actSave->setStatusTip(tr("Save changes"));
actSaveAs->setText(tr("Save &as..."));
actSaveAs->setStatusTip(tr("Save changes in a new file"));
actExport->setText(tr("&Export to raster..."));
actExport->setStatusTip(tr("Save the flowchart in a raster picture format"));
actExportSVG->setText(tr("&Export to SVG..."));
actExportSVG->setStatusTip(tr("Save the flowchart in a vector picture format"));
actPrint->setText(tr("&Print..."));
actPrint->setStatusTip(tr("To print"));
actNew->setText(tr("&New"));
actNew->setStatusTip(tr("Create a new project"));
actUndo->setText(tr("&Undo"));
actUndo->setStatusTip(tr("Undo the last operation"));
actRedo->setText(tr("&Redo"));
actRedo->setStatusTip(tr("Restore the last undone action"));
actCut->setText(tr("Cu&t"));
actCut->setStatusTip(tr("Cut the current selection"));
actCopy->setText(tr("&Copy"));
actCopy->setStatusTip(tr("Copy the current selection"));
actPaste->setText(tr("&Paste"));
actPaste->setStatusTip(tr("Paste"));
actDelete->setText(tr("&Delete"));
actDelete->setStatusTip(tr("Delete the current selection"));
actHelp->setText(tr("&Help"));
actHelp->setStatusTip(tr("Toggle Help window"));
actAbout->setText(tr("&About"));
actAbout->setStatusTip(tr("Information about authors"));
actAboutQt->setText(tr("About &Qt"));
actAboutQt->setStatusTip(tr("Information about Qt"));
actTools->setText(tr("&Tools"));
actTools->setStatusTip(tr("Toggle the tool panel"));
actCode->setText(tr("&Source code"));
actCode->setStatusTip(tr("Toggle the source code panel"));
actExit->setShortcut(tr("Alt+X"));
actOpen->setShortcut(tr("Ctrl+O"));
actSave->setShortcut(tr("Ctrl+S"));
actNew->setShortcut(tr("Ctrl+N"));
actUndo->setShortcut(tr("Ctrl+Z"));

```

```

actRedo->setShortcut(tr("Ctrl+Y"));
actCut->setShortcut(tr("Ctrl+X"));
actCopy->setShortcut(tr("Ctrl+C"));
actPaste->setShortcut(tr("Ctrl+V"));
actDelete->setShortcut(tr("Del"));
actHelp->setShortcut(tr("F1"));
actPrint->setShortcut(tr("Ctrl+P"));
actTools->setShortcut(tr("F2"));
actCode->setShortcut(tr("F3"));
menuFile->setTitle(tr("&File"));
menuEdit->setTitle(tr("&Edit"));
menuHelp->setTitle(tr("&Help"));
menuWindow->setTitle(tr("&View"));
menuLanguage->setTitle(tr("&Language"));
toolBar->setWindowTitle(tr("Standard"));
dockCode->setWindowTitle(tr("Source code"));

slotReloadGenerators();
codeLabel->setText(tr("&Select programming language:"));
if (!fileName.isEmpty())
    setWindowTitle(tr("%1 - Algorithm Flowchart Editor").arg(fileName));
else
    setWindowTitle(tr("Algorithm Flowchart Editor"));
helpWindow->setWindowTitle(tr("Help window"));
helpWindow->textBrowser->setSearchPaths(QStringList() << "/help/" + QLocale().name()
<< "/help/en_US");
#ifdef Q_WS_X11 || defined(Q_OS_LINUX)
    helpWindow->textBrowser->setSearchPaths(QStringList()
QString(PROGRAM_DATA_DIR) + "help/" + QLocale().name()
QString(PROGRAM_DATA_DIR) + "help/en_US");
#endif
helpWindow->textBrowser->reload();
}
void MainWindow::createMenu()
{
    menuFile = menuBar()->addMenu("");
    menuFile->addAction(actNew);
    menuFile->addAction(actOpen);
    menuFile->addSeparator();
    menuFile->addAction(actSave);
    menuFile->addAction(actSaveAs);
    menuFile->addSeparator();
    menuFile->addAction(actExport);
    menuFile->addAction(actExportSVG);
    menuFile->addSeparator();
    menuFile->addAction(actPrint);
    menuFile->addSeparator();
    menuFile->addSeparator();
    menuFile->addAction(actExit);
    actAbout->isChecked();
    menuEdit = menuBar()->addMenu("");
    menuEdit->addAction(actUndo);
    menuEdit->addAction(actRedo);

```

```

menuEdit->addSeparator();
menuEdit->addAction(actCut);
menuEdit->addAction(actCopy);
menuEdit->addAction(actPaste);
menuEdit->addSeparator();
menuEdit->addAction(actDelete);
menuWindow = menuBar()->addMenu("");
menuWindow->addAction(actTools);
menuWindow->addAction(actCode);
menuWindow->addSeparator();
menuLanguage = menuWindow->addMenu(tr("&Language"));
for (int i = 0; i < actLanguages.size(); ++i) {
    menuLanguage->addAction(actLanguages[i]);
}
menuHelp = menuBar()->addMenu("");
menuHelp->addAction(actHelp);
menuHelp->addSeparator();
menuHelp->addAction(actAbout);
menuHelp->addAction(actAboutQt);
}
void MainWindow::createToolBar()
{
    toolBar = addToolBar("");
    toolBar->setObjectName("standard_toolbar");
    toolBar->setIconSize(QSize(32,32));
    toolBar->setToolButtonStyle(Qt::ToolButtonTextUnderIcon);
    toolBar->addAction(actNew);
    toolBar->addAction(actOpen);
    toolBar->addAction(actSave);
    toolBar->addSeparator();
    toolBar->addAction(actUndo);
    toolBar->addAction(actRedo);
    toolBar->addSeparator();
    toolBar->addAction(actCut);
    toolBar->addAction(actCopy);
    toolBar->addAction(actPaste);
    toolBar->addSeparator();
    toolBar->addAction(actHelp);
    toolBar->addSeparator();
    toolBar->addAction(actTools);
    toolBar->addAction(actCode);
}
void MainWindow::docToolsVisibilityChanged(bool visible)
{
    actTools->setChecked(visible);
}
void MainWindow::docCodeVisibilityChanged(bool visible)
{
    actCode->setChecked(visible);
}
void MainWindow::createActions()
{
    actExit = new QAction(QIcon(":/images/exit.png"), "", this);

```

```

actOpen = new QAction(QIcon(":/images/open_document_32_h.png"), "", this);
actNew = new QAction(QIcon(":/images/new_document_32_h.png"), "", this);
actSave = new QAction(QIcon(":/images/save_32_h.png"), "", this);
actSaveAs = new QAction(this);
//actUndo = new QAction(QIcon(":/images/undo_32_h.png"), "", this);
//actRedo = new QAction(QIcon(":/images/redo_32_h.png"), "", this);
actUndo = new QAction(QIcon(":/images/restart-3.png"), "", this);
actRedo = new QAction(QIcon(":/images/restart-4.png"), "", this);
actCut = new QAction(QIcon(":/images/cut_clipboard_32_h.png"), "", this);
actCopy = new QAction(QIcon(":/images/copy_clipboard_32_h.png"), "", this);
actPaste = new QAction(QIcon(":/images/paste_clipboard_32_h.png"), "", this);
actDelete = new QAction(QIcon(":/images/delete_x_32_h.png"), "", this);
actExport = new QAction(this);
actExportSVG = new QAction(this);
actHelp = new QAction(QIcon(":/images/help-icon.png"), "", this);
actAbout = new QAction(this);
actAboutQt = new QAction(this);
actPrint = new QAction(QIcon(":/images/print_32_h.png"), "", this);
actTools = new QAction(QIcon(":/images/toolbar.png"), "", this);
actCode = new QAction(QIcon(":/images/source-code.png"), "", this);
connect(actExit, SIGNAL(triggered()), this, SLOT(close()));
connect(actNew, SIGNAL(triggered()), this, SLOT(slotFileNew()));
connect(actOpen, SIGNAL(triggered()), this, SLOT(slotFileOpen()));
connect(actSave, SIGNAL(triggered()), this, SLOT(slotFileSave()));
connect(actSaveAs, SIGNAL(triggered()), this, SLOT(slotFileSaveAs()));
connect(actExport, SIGNAL(triggered()), this, SLOT(slotFileExport()));
connect(actExportSVG, SIGNAL(triggered()), this, SLOT(slotFileExportSVG()));
connect(actPrint, SIGNAL(triggered()), this, SLOT(slotFilePrint()));
connect(actCut, SIGNAL(triggered()), this, SLOT(slotEditCut()));
connect(actCopy, SIGNAL(triggered()), this, SLOT(slotEditCopy()));
connect(actPaste, SIGNAL(triggered()), this, SLOT(slotEditPaste()));
connect(actDelete, SIGNAL(triggered()), this, SLOT(slotEditDelete()));
// connect(actHelp, SIGNAL(triggered()), this, SLOT(slotHelpHelp()));
// connect(actAbout, SIGNAL(triggered()), this, SLOT(slotHelpAbout()));
// connect(actAboutQt, SIGNAL(triggered()), this, SLOT(slotHelpAboutQt()));
/* ... */
// connect(actTools, SIGNAL(triggered()), this, SLOT(slotTools()));
QHash<QString, QString> avlLangs = enumLanguages();
QList<QString> locales = avlLangs.keys();
for (int i = 0; i < locales.size(); ++i) {
    QAction *act = new QAction(this);
    act->setText(avlLangs[locales[i]]);
    act->setData(locales[i]);
    actLanguages.append(act);
    connect(act, SIGNAL(triggered()), this, SLOT(slotChangeLanguage()));
}
}
MainWindow::~MainWindow()
{
}
bool MainWindow::okToContinue()
{
    int r;

```



```

if (!isSaved) {
    r = QMessageBox::warning(this,
                             tr("Afcce"), tr("There are unsaved changes. Do you really want to close"),
                             QMessageBox::Yes | QMessageBox::Default,
                             QMessageBox::No | QMessageBox::Escape);

    if (r != QMessageBox::Yes) {
        return false;
    }
}
return true;
}

void MainWindow::slotFileOpen()
{
    QString fn = QFileDialog::getOpenFileName ( this,
                                                tr("Select a file to open"), "", tr("Algorithm flowcharts
(*.afc)"));
    if(!fn.isEmpty())
    {

        if (!isSaved)
            if(QMessageBox::warning(this, tr("Unsaved changes"), tr("You are about to open
another document. It will discard all unsaved changes in the current document."),
                                   QMessageBox::Ok | QMessageBox::Cancel) != QMessageBox::Ok)
                return;
        slotOpenDocument(fn);
    }
}

void MainWindow::slotOpenDocument(const QString &fn) {
    emit documentUnloaded();
    fileName = fn;
    setWindowTitle(tr("%1 - Algorithm Flowchart Editor").arg(fileName));
    QFile xml(fileName);
    if (xml.exists())
    {
        xml.open(QIODevice::ReadOnly | QIODevice::Text);
        QDomDocument doc;
        if (doc.setContent(&xml, false))
        {
            document()->root()->setXmlNode(doc.firstChildElement());
            document()->setZoom(1);
        }
    }
    emit documentLoaded();
}

void MainWindow::slotFilePrint()
{
    QPrinter printer(QPrinter::HighResolution);
    QPrintDialog pd(&printer, this);
    if (pd.exec() == QDialog::Accepted)
    {

```

```

double oldZoom = document()->zoom();
document()->setZoom(1);
document()->setStatus(QFlowChart::Display);
QBlock *r = document()->root();
r->adjustSize(1);
r->adjustPosition(0,0);
QRect page = printer.pageRect();
double z = page.width() / (double) r->width;
if (r->height * z > page.height())
{
    z = page.height() / (double) r->height;
}
if (z > (printer.resolution()/96.0)) z = printer.resolution()/96.0;
document()->setZoom(z);
r->adjustSize(z);
r->adjustPosition(0, 0);
QPainter canvas;
canvas.begin(&printer);
document()->paintTo(&canvas);
canvas.end();
document()->setZoom(oldZoom);
document()->setStatus(QFlowChart::Selectable);
}
}
void MainWindow::slotFileNew()
{
    QProcess::startDetached(QApplication::applicationFilePath());
}
void MainWindow::slotFileSave()
{
    if (fileName.isEmpty())
    {
        slotFileSaveAs();
    }
    else
    {
        QDomDocument doc = document()->document();
        QString xmlString = doc.toString(2);
        QFile xml(fileName);
        xml.open(QIODevice::ReadWrite | QIODevice::Text | QIODevice::Truncate);
        QTextStream stream(&xml);
        stream.setCodec(QTextCodec::codecForName("utf-8"));
        stream << xmlString;
        xml.close();
        emit documentSaved();
    }
}

void MainWindow::slotFileSaveAs()
{
    QString fn = QFileDialog::getSaveFileName(this, tr("Select a file to save"), "",
tr("Algorithm flowcharts (*.afc)"));
    if (!fn.isEmpty())

```

```

{
    if(fn.right(4).toLower() != ".afc") fn += ".afc";
    fileName = fn;
    setWindowTitle(tr("%1 - Algorithm Flowchart Editor").arg(fileName));
    slotFileSave();
}
}
void MainWindow::slotDocumentSaved() {
    isSaved = true;
}

void MainWindow::slotDocumentChanged() {
    isSaved = false;
}
void MainWindow::slotDocumentLoaded() {
    isSaved = true;
}
void MainWindow::slotChangeLanguage()
{
    QAction * action = (QAction *)sender();
    QString localeName = action->data().toString();
    setApplicationLocale(localeName);
    retranslateUi();
    QSettings settings("afce", "application");
    settings.setValue("locale", localeName);
}
void MainWindow::slotReloadGenerators()
{
    int i = codeLanguage->currentIndex();
    codeLanguage->clear();
    QDir gd("generators:");
    QStringList gens = gd.entryList(QStringList() << "*.json", QDir::Files, QDir::Name);
    for (int g = 0; g < gens.size(); ++g) {
        QFile f(gd.absoluteFilePath(gens[g]));
        QString lang_name = QFileInfo(f.fileName()).baseName();
        if(f.open(QIODevice::ReadOnly|QIODevice::Text)) {
            QJsonDocument json = QJsonDocument::fromJson(f.readAll());
            f.close();
            QJsonObject obj = json.object();
            QString loc = QLocale().name();
            if(obj.contains("name")) {
                QJsonObject jn = obj.value("name").toObject();
                if(jn.contains(loc)) {
                    lang_name = jn.value(loc).toString();
                }
                else if (jn.contains("en_US")) {
                    lang_name = jn.value("en_US").toString();
                }
            }
            codeLanguage->addItem(lang_name, QFileInfo(f.fileName()).baseName());
        }
    }
    if (i!=-1)

```

```

        codeLanguage->setCurrentIndex(i);
    else
        codeLanguage->setCurrentIndex(0);
}
void MainWindow::slotFileExport()
{
    QString filter = getWriteFormatFilter();
    QString sf = getFilterFor("png");
    QString fn = QFileDialog::getSaveFileName(this, tr("Select a file to export"), "", filter,
&sf);
    if(!fn.isEmpty())
    {
        qDebug() << "Selected filter: " << sf;
        QRegExp rx("\\(\\*([^\|]+)\|)\$");
        QStringList masks;
        if(rx.indexIn(sf) != -1) {
            // each filter may content several masks splitted by a spaces
            masks = rx.cap(1).split(" ", QString::SkipEmptyParts);
            bool matches = false;
            for(int i = 0; i < masks.size(); ++i) {
                QString ex = masks.at(i).toLowerCase();
                if(fn.toLowerCase().endsWith(ex)) {
                    matches = true;
                    break;
                }
            }
        }
        // if no extension or a wrong extension is added then the correct extension will be
appended
        if(!matches && !masks.empty()) {
            fn += masks.first().toLowerCase();
        }
    }
    double oldZoom = document()->zoom();
    document()->setZoom(1);
    document()->setStatus(QFlowChart::Display);
    QBlock *r = document()->root();
    r->adjustSize(1);
    r->adjustPosition(0,0);
    QImage img(r->width, r->height, QImage::Format_ARGB32_Premultiplied);
    img.fill(0);
    QPainter canvas(&img);
    canvas.setRenderHint(QPainter::Antialiasing);
    document()->paintTo(&canvas);
    img.save(fn);
    document()->setZoom(oldZoom);
    document()->setStatus(QFlowChart::Selectable);
}
}
void MainWindow::slotFileExportSVG()
{
    QString filter = getWriteFormatFilter();
    QString fn = QFileDialog::getSaveFileName(this, tr("Select a file to export"), "",
getFilterFor("svg"));

```

```

if(!fn.isEmpty())
{
    if(fn.right(4).toLowerCase() != ".svg") fn += ".svg";
    double oldZoom = document()->zoom();
    document()->setZoom(1);
    document()->setStatus(QFlowChart::Display);
    QBlock *r = document()->root();
    r->adjustSize(1);
    r->adjustPosition(0,0);
    QSvgGenerator svg;
    svg.setSize(QSize(r->width, r->height));
    svg.setResolution(90);
    svg.setFileName(fn);
    QPainter canvas(&svg);
    canvas.setRenderHint(QPainter::Antialiasing);
    r->paint(&canvas, true);
    document()->setZoom(oldZoom);
    document()->setStatus(QFlowChart::Selectable);
}
}
void MainWindow::slotEditCut()
{
    slotEditCopy();
    slotEditDelete();
}
void MainWindow::slotEditCopy()
{
    if(document())
    {
        if(document()->activeBlock())
        {
            QDomDocument doc("AFC"); // do not localize!
            QDomElement block = document()->activeBlock()->xmlNode(doc);
            if (document()->activeBlock()->isBranch)
            {
                QDomElement alg = doc.createElement("algorithm");
                alg.appendChild(block);
                doc.appendChild(alg);
            }
            else
            {
                if(block.nodeName() != "algorithm")
                {
                    QDomElement alg = doc.createElement("algorithm");
                    QDomElement branch = doc.createElement("branch");
                    alg.appendChild(branch);
                    branch.appendChild(block);
                    doc.appendChild(alg);
                }
                else
                {
                    doc.appendChild(block);
                }
            }
        }
        QClipboard *clipbrd = QApplication::clipboard();
    }
}

```

```

        clipbrd->setText(doc.toString(2));
        updateActions();
    }
}
}
void MainWindow::slotEditPaste()
{
    if(document())
    {
        QClipboard *clipbrd = QApplication::clipboard();
        document()->setBuffer(clipbrd->text());
        if(!document()->buffer().isEmpty())
        {
            document()->setStatus(QFlowChart::Insertion);
            document()->setMultiInsert(false);
        }
    }
}
void MainWindow::slotEditDelete()
{
    if(document())
    {
        if(document()->status() == QFlowChart::Selectable)
        {
            document()->deleteActiveBlock();
        }
    }
}

```