

Циклова комісія Комп'ютерна інженерія

Плани
самостійного вивчення тем
з дисципліни
«Технології (Програмування)»

Розробив: _____ О.В.Старосельцева

Розглянуто та схвалено

на засіданні предметної (циклової)

комісії комп'ютерної інженерії

Протокол № 1 від 01.09.2022 р.

Голова комісії _____ О.В. Старосельцева

Зміст

№	Тема для самостійного вивчення
1	Комп'ютерні програми і мови програмування
2	Історія створення та перспективи розвитку мови програмування C/C++
3	Операції над числовими типами даних: бінарні побітові операції
4	Стандартні функції обробки рядків символів.
5	Обробка статичних масивів даних з використанням вказівників
6	Вказівники на структури. Передача структур функціям.
7	Шаблон auto_ptr. Додаткові функції стандартного введення – виведення.

План самостійного вивчення теми № 1

з навчальної дисципліни Технології (Програмування)

- Тема** Комп'ютерні програми і мови програмування
- Мета** Ввести поняття комп'ютерної програми, програмної логіки та інтерфейсу користувача. Охарактеризувати мови програмування. Дати поняття та призначення компілятора та інтерпретатора.

знати:

- поняття комп'ютерної програми, програмної логіки;
- що називається інтерфейсом користувача;
- поняття мови програмування;
- відміну машинно-орієнтованих мов програмування від мов високого рівня;
- поняття інтерпретатора та компілятора.

вміти:

- виділяти вхідні, вихідні та проміжні дані задачі;
- характеризувати компоненти мови програмування.

Час – 2 години

Навчальні питання:

- 1 Комп'ютерні програми
- 2 Мови програмування

Методичні рекомендації

Самостійно вивчаючи запропоновану тему, студент повинен засвоїти такі основні положення:

По-перше Програма – це набір команд (вказівок, інструкцій), призначений для виконання комп'ютером у певній послідовності.

- По-друге* Мова, яка використовується для запису алгоритмів, призначених для виконання комп'ютером, називається мовою програмування.
- По-третє* Кожна мова програмування має такі компоненти:
- алфавіт – множину символів, з яких можна утворювати слова та речення цієї мови;
 - словник – набір спеціальних (зарезервованих, ключових) слів;
 - синтаксис – правила складання та запису мовних конструкцій (не словникових слів і речень);
 - семантику – встановлене однозначне тлумачення мовних конструкцій, правил їх виконання.
- По-четверте* Для перекладу програм на машинну мову створені та використовуються спеціальні програми – **компілятори**. Для деяких мов програмування створено інші спеціальні програми – **інтерпретатори**.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс]
[/http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html](http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html)

Навчальні матеріали до самостійного вивчення теми

1 Комп'ютерні програми

Ви знаєте, що комп'ютер працює під управлінням програмного забезпечення, яке складається з комп'ютерних програм (далі – програм) різноманітного призначення. Працюючи з комп'ютером, ви використовували

текстовий процесор і графічний редактор, програми-архіватори й антивірусні програми, табличний процесор і редактор комп'ютерних презентацій, навчальні та контролюючі програми, ігрові програми та багато інших.

Програма – це набір команд (вказівок, інструкцій), призначений для виконання комп'ютером у певній послідовності.

Програми складаються для виконання комп'ютером алгоритмів. Ці алгоритми утворюють логіку програми (програмну логіку). У процесі своєї роботи програма опрацьовує дані. Дані, які вводять до програми безпосередньо користувач програми або програма їх отримує від певного пристрою (наприклад, від датчика температури), або від іншої програми, або з іншого джерела (наприклад, з текстового файлу) називаються **вхідними (початковими)** даними. Деякі програми працюють без вхідних даних. Дані, отримання яких є метою використання програми, називаються **вихідними (результуючими)** даними. Під час виконання програми утворюються та опрацьовуються й інші дані, які називаються проміжними даними.

Більшість сучасних програм у процесі своєї роботи надають користувачу певний набір засобів для його взаємодії з програмою та пристроями. До цих засобів належать засоби керування (кнопки, меню та ін.), засоби введення даних (поля, лічильники та ін.), засоби виведення даних (написи, поля і т. д.) та ін. Сукупність таких засобів, а також методів їх використання утворюють **інтерфейс користувача**.

2 Мови програмування

Складаючи алгоритми, призначені для виконання людиною, користуються мовою спілкування людей: українською, російською, англійською, німецькою тощо. Але для алгоритмів, які повинен виконувати автоматичний пристрій (зокрема, комп'ютер), мова спілкування людей складна, має неоднозначні конструкції (наприклад, слова-омоніми). Тому для

запису алгоритмів, які призначені для виконання автоматичними пристроями, розробляють і використовують спеціальні мови – мови програмування.

Мова, яка використовується для запису алгоритмів, призначених для виконання комп'ютером, називається **мовою програмування**.

Кожна мова програмування має такі компоненти:

- алфавіт – множину символів, з яких можна утворювати слова та речення цієї мови;
- словник – набір спеціальних (зарезервованих, ключових) слів;
- синтаксис – правила складання та запису мовних конструкцій (не словникових слів і речень);
- семантику – встановлене однозначне тлумачення мовних конструкцій, правил їх виконання.

Використання символів, що не входять до алфавіту, неправильне написання словникових слів, порушення синтаксичних правил призводить до неможливості виконання комп'ютером відповідної команди. Такі порушення називаються **синтаксичними помилками**.

За останні 70 років створено близько трьох тисяч різних мов програмування. Деякі з них уже вийшли з користування, для деяких постійно з'являються досконаліші версії, кожна наступна з яких зручніша для складання програм і має ширші можливості, постійно створюються нові мови програмування. Деякі мови програмування використовуються для складання програм для розв'язування задач з різних галузей науки, техніки, виробництва, сфери побуту та ін. (наприклад, Delphi, C++, C#, Java), а деякі створені спеціально для складання програм для розв'язування спеціального кола задач (наприклад, Prolog (англ. Programming in Logic – програмування в логіці), Lisp (англ. List Processing – опрацювання списків)).

Процесор комп'ютера може виконувати команди, подані тільки машинною мовою. **Машинна мова** – це мова програмування, в якій команди подаються як послідовності двійкових кодів. Машинна мова програмування орієнтована на процесори конкретної архітектури, тобто машинні мови для

різних процесорів можуть відрізнятися одна від одної. Для виконання процесором програм, написаних не машинною мовою програмування, їх потрібно спочатку перекласти на машинну мову і лише потім виконати.

Різні типи процесорів мають різний набір команд. Якщо мова програмування орієнтована на конкретний тип процесора і враховує його особливості, то він називається мовою програмування низького рівня. Мовою найнижчого рівня є мова асемблера, який просто представляє кожен команду машинної коду у вигляді спеціальних символічних позначень, які називаються мнемоніками. За допомогою мов низького рівня створюються дуже ефективні і компактні програми, оскільки розробник дістає доступ до всіх можливостей процесора. Оскільки набори інструкцій для різних моделей процесорів теж різні, то кожній моделі процесора відповідає своя мова асемблера, і написана на ньому програма може бути використана тільки в цьому середовищі. Подібні мови застосовують для написання невеликих системних застосувань, драйверів пристроїв тощо.

Мови програмування високого рівня не враховують особливості конкретної комп'ютерної архітектури, тому створювані програми на рівні початкових текстів легко переносяться на інші платформи, якщо для них створені відповідні транслятори. Розробка програм на мовах високого рівня набагато простіше, ніж на машинних мовах.

Мовами високого рівня є:

1 Фортран – перша компільована мова, створена в 50-і роки 20 століть. У ній були реалізований ряд найважливіших понять програмування. Для цієї мови було створено величезну кількість бібліотек, починаючи від статистичних комплексів і закінчуючи управлінням супутниками, тому вона продовжує використовуватися в багатьох організаціях.

2 Кобол – компільована мова для економічних розрахунків і вирішення бізнес-задач, розроблена на початку 60-х років. У Коболі були реалізовані дуже могутні засоби роботи з великими об'ємами даних, що зберігаються на зовнішніх носіях.

3 Паскаль – створений в кінці 70-х років швейцарським математиком Ніклаусом Віртом спеціально для навчання програмуванню. Вона дозволяє виробити алгоритмічне мислення, будувати коротку, добре читану програму, демонструвати основні прийоми алгоритмізації, вона також добре підходить для реалізації крупних проектів.

4 Бейсік – створювався в 60-х роках також для навчання програмуванню. Для нього є і компілятори і інтерпретатори, є одним з найпопулярніших мов програмування.

5 Сі – був створений в 70-х роках, спочатку не розглядався як масова мова програмування. Він планувався для заміни асемблера, щоб мати можливість створювати такі ж ефективні і короткі програми, але не залежати від конкретного процесора. Він багато в чому схожий на Паскаль і має додаткові можливості для роботи з пам'яттю. На ньому написано багато прикладних і системних програм, а також операційну систему Unix.

6 C++ – об'єктно-орієнтоване розширення мови Сі, створене Бьярном Страуструпом в 1980 р.

7 Java – мова, яка була створена компанією Sun на початку 90-х років на основі C++. Вона покликана спростити розробку додатків на C++ шляхом виключення з нього низькорівневих можливостей. Головна особливість мови – це те, що він компілюється не в машинний код, а в незалежний байт-код (кожна команда займає один байт). Цей код може виконуватися за допомогою інтерпретатора – віртуальної Java-машини (JVM).

Для перекладу програм на машинну мову створені та використовуються спеціальні програми – **компілятори**. Ці програми аналізують весь текст програми на наявність синтаксичних помилок, і якщо такі помилки відсутні, перекладають текст програми на машинну мову, формуючи машинний код програми. Цей код, залежно від режиму роботи компілятора, або зберігається в пам'яті комп'ютера, або записується на диск у вигляді виконуваного файлу (наприклад, exe-файлу). Після отримання виконуваного файлу його можна відправити на виконання процесором. При цьому сама програма-компілятор

вже не використовується. Тому виконуваний файл може використовуватися й на тих комп'ютерах, де програма-компілятор відсутня. За наявності в програмі синтаксичних помилок, компілятор або зупиняється на першій з них і виводить на екран повідомлення про неї, або аналізує програму до кінця та виводить на екран загальний список повідомлень про наявні помилки. Після цього потрібно виправити всі синтаксичні помилки і розпочати процес компіляції знову.

Для деяких мов програмування створено інші спеціальні програми – *інтерпретатори*. Ці програми не створюють виконуваних файлів, а аналізують програму покомандно й одразу ж ці команди виконують. Тому виконати програму, яка інтерпретується, а не компілюється, можна лише на тому комп'ютері, де встановлена відповідна програма-інтерпретатор. Для деяких сучасних мов програмування використовують комбінацію компіляції й інтерпретації. Спочатку програма компілюється в деякий проміжний код (не машинний), після чого інтерпретується спеціальною програмою, написаною для цього коду

Питання для самоконтролю знань студентів

- 1 Що таке комп'ютерна програма?
- 2 Що таке програмна логіка й інтерфейс користувача?
- 3 Які дані називають вхідними, вихідними, проміжними?
- 4 Що таке мова програмування?
- 5 Назвіть компоненти, з яких складається мова програмування.
- 6 Опишіть кожний компонент мови програмування.
- 7 Що таке синтаксична помилка?
- 8 Що таке машинна мова програмування?
- 9 Який вигляд мають команди в цій мові програмування?
- 10 Які програми називаються компіляторами? Опишіть загальний принцип їх роботи.

11 Які програми називаються інтерпретаторами? Опишіть загальний принцип їх роботи.

План самостійного вивчення теми № 2

з навчальної дисципліни Технології (Програмування)

Тема Історія створення та перспективи розвитку мови програмування C/C++

Мета Ознайомлення студентів з передумовами створення мови програмування C, оглядова характеристика структурного, об'єктно-орієнтованого та узагальненого підходів до програмування, огляд перспектив розвитку C++

знати:

- передумови створення мови програмування C;
- характеристика структурного, об'єктно-орієнтованого та узагальненого підходів до програмування;
- перспективи розвитку C++.

Час – 4 години

Навчальні питання:

- 1 Передумови створення мови C.
- 2 Філософія програмування, закладена в мові C++
 - 2.1 Процедурне програмування
 - 2.2 Об'єктно-орієнтоване програмування
 - 2.3 Узагальнене програмування
- 3 Мова C++. Перспективи розвитку.

Методичні рекомендації

Самостійно вивчаючи запропоновану тему, студент повинен засвоїти такі основні положення:

По-перше Мова C була створена як мова, яка б поєднувала ефективність і можливість доступу до апаратних засобів, наявних в мовах низького рівня, з більше загальним характером і переносимістю, властивих мовам високого рівня.

По-друге

Мова С, як і більшість основних мов програмування нашого часу, є процедурною. Це означає, що основна увага в ньому приділяється алгоритмам. Теоретично процедурне програмування полягає в тім, що спочатку визначається послідовність дій, що повинна бути виконана комп'ютером, а потім ці дії реалізуються за допомогою мови програмування.

По-третє

Об'єктно-орієнтований підхід до розробки програми полягає в тому, що спочатку розробляються класи, що точно представляють ті речі, з якими має справу програма.

У мові С++ клас є специфікацією, що описує таку нову форму даних, а об'єкт - конкретною структурою даних, створеної відповідно до цієї специфікації.

По-четверте

Узагальнене програмування - це ще одна парадигма програмування, підтримувана мовою С++. Воно має загальну з ООП мету - спростити повторне використання кодів програм і методів абстрагування загальних понять. Однак, у той час як в ООП основну увагу приділяється даним, в узагальненому програмуванні упор робиться на алгоритми. Це забезпечується за допомогою шаблонів мови С++.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс]
[/http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html](http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html)

Навчальні матеріали до самостійного вивчення теми

ВСТУП

В останні кілька десятиліть комп'ютерна технологія розвивалася разючими темпами. У цей час переносний комп'ютер може зберігати більше

інформації й робити обчислення швидше, ніж більша ЕВМТРИДЦАТЬ років тому. (Далеко не всі програмісти можуть згадати, як доводилося носити більші колоди перфокарт, які вводилися в потужні, що заповнюють целую кімнату обчислювальні системи із грандіозним (по тимі часам) обсягом пам'яті 100 Кб. Тепер цього обсягу недостатньо для запуску гарної гри на персональному комп'ютері.) Мови програмування також перетерпіли значну еволюцію. Зміни, можливо, не були такими вpečаляючими, однак вони були дуже важливими. Могутніші комп'ютери породжували більші й складні програми, які, у свою чергу, з нові проблеми в області керування програмами, а також їхнього супроводу.

В 70-і роки такі мови програмування, як C и Pascal, допомогли увійти в еру структурного програмування, що принесло порядок у ту область, що сильно мала потребу в цьому. Мова C надав у розпорядження програміста інструменти, необхідні для структурного програмування, а також забезпечив створення компактних, швидко працюючих програм і можливість адресації апаратних засобів, наприклад, можливість керування портами зв'язку й накопичувачами на магнітних дисках. Ці якості допомогли мові C стати пануючою мовою програмування в 80-і роки. Разом з тим у ці роки з'явилася нова модель програмування: об'єктно-орієнтоване програмування, або ООП, втілене в таких мовах, як SmallTalk й C++. Давайте розглянемо тепер мову C и об'єктно-орієнтоване програмування докладніше.

1 Передумови створення мови C.

На початку 70-х років Деніс Рітчі (Dennis Ritchie) з компанії Bell Laboratories з розробкою операційної системи UNIX. (Операційна система - це сукупність програм, що управляють ресурсами комп'ютера і його взаємодією з користувачами. Наприклад, саме операційна система виводить на екран системні повідомлення-підказки й управляє виконанням ваших програм.) Для виконання цієї роботи Рітчі мав потребу в такій мові програмування, що був

би коротким, а також міг би забезпечувати ефективне керування апаратними засобами й створення компактних, швидко працюючих програм. Традиційно такі потреби програмістів задовольняв мову асемблера, що тісно пов'язаний із внутрішньою машинною мовою комп'ютера. Однак мова асемблера - це мова низького рівня, тобто він прив'язаний до певного типу процесора (або комп'ютера). Тому якщо програму мовою асемблера необхідно перенести на комп'ютер іншого типу, те її доводиться переписувати заново на іншій мові асемблера. Це мало-мало схоже на те, як якби при покупці нового автомобіля ви щораз виявляли, що конструктори вирішили змінити розташування й призначення органів керування, змушуючи вас переучуватися водінню. Операційна система UNIX призначалася для роботи на різноманітних типах комп'ютерів (або платформах). А це припускало використання мови високого рівня. Мова високого рівня орієнтований на рішення завдань, а не на конкретне апаратне забезпечення. Спеціальні програми, які називаються компіляторами, транслюють програму мовою високого рівня в програму внутрішньою мовою конкретного комп'ютера. Таким чином, використовуючи окремий компілятор для кожної платформи, ту саму програму мовою високого рівня можна виконувати на різних платформах. Рітчи мав потребу в мові, який би поєднував ефективність і можливість доступу до апаратних засобів, наявні в мови низького рівня, з більше загальним характером і переносимістю, властивій мові високого рівня. Тому на основі більше старих мов, що були в той час, програмування Рітчи розробив мову C.

Мова C була створена в 1972 р. співробітником фірми Bell Laboratories Деннісом Рітчи, коли він і Кен Томпсон займалися розробкою ОС UNIX. Проте мова Cі не виникла сама собою. Вона з'явилася від розробленої Томпсоном мови B. Важливим моментом є те, що мова Cі була створена як інструмент для працюючих програмістів, тому головна мета розробки цієї мови програмування полягає в тому, щоб її застосування було корисним при розробці різних програм.

Більшість мов програмування націлена на те, щоб бути корисними, але при цьому їх розробники переслідували і інші цілі. Головна мета розробки мови Pascal, наприклад, полягала в забезпеченні міцної основи для викладання правильних принципів програмування. З другого боку, мова програмування Basic розроблялася так, щоб бути схожою на англійську мову, тому її легко могли вивчати студенти, не знайомі з комп'ютерами. Проте всі ці цілі не завжди узгоджуються з повсякденною практичною користю. А ось мета розробки мови програмування Сі, що полягає в створенні інструментального засобу, призначеного для програмістів, привела до того, що мова Сі стала однією з кращих мов програмування нашого часу.

Якщо судити по числу зарезервованих слів, то Сі – це мала мова, і тому існує думка, що її порівняно легко вивчити. В той же час Сі є виключно ефективною мовою. Використовуючи невелике число конструкцій Сі, можна будувати великі і складні програмні системи і вирішувати важкі задачі. Сама мова Сі і стандартні бібліотеки, що поставляються з більшістю реалізацій мови, є базовими засобами для створення програмних компонентів, що повторно використовуються, які можна застосовувати в багатьох додатках.

В порівнянні з менш могутніми і більш простими мовами Сі надає програмісту високий рівень гнучкості, покладаючи на нього більш високу відповідальність. Збільшення гнучкості може сприяти побудові компактного і ефективно виконуваного коду, але може і привести до заплутаної і важко читаної програми. Тому програміст повинен відповідально відноситися до написання на Сі простих і технологічних програм. Для ефективного використання Сі потрібні навчання, досвід і навички.

Існує ряд причин, по яких мова Сі стала вельми популярною останніми роками. Основна з них полягає у високій швидкості виконання одержуваного коду і його компактності, що особливо цінно для професійних програм.

Для мови Сі характерна достатньо висока ефективність. В ній використовуються можливості сучасних комп'ютерів. Фактично мова Сі володіє такими прекрасними можливостями управління, які звичайно

асоціюються з мовою асемблера. Програми можна за бажанням налаштувати або на максимальну швидкість виконання, або на економне (ефективне) використання доступної пам'яті.

Мова Сі є кросплатформеною мовою. Це означає, що програми написані для однієї ОС, можуть виконуватися в інших системах з невеликими змінами (або взагалі без таких).

Якщо модифікація програми необхідна, то часто вона може бути виконана простою заміною декількох записів в заголовному файлі для основного модуля. Існують компілятори мови Сі приблизно для 40 ОС, починаючи від систем для 8-бітових мікропроцесорів і закінчуючи системами для суперкомп'ютерів Cray.

Багато компіляторів і інтерпретаторів для інших мов програмування таких як Fortran, APL, Pascal, LISP, Logo і Basic, написані на мові Сі. Програми Сі використовувалися для вирішення фізичних і інженерних задач, і навіть для створення спеціальних мультиплікаційних ефектів в кінофільмах.

Недоліки. Мова програмування Сі має деякі недоліки. Наприклад, свобода в написанні виразів в мові Сі вимагає великої відповідальності.

Компактність мови Сі в поєднанні з великою кількістю операторів дає можливість створювати програмний код, розуміння якого надзвичайно складно. Ніхто не примушує програміста створювати незрозумілі програми, але всі можливості для цього є.

2 Філософія програмування, закладена в мові C++

2.1 Процедурне програмування

Оскільки мова C++ вносить у мову C нову філософію програмування, нам варто спочатку розглянути більше стару філософію мови C. Взагалі, мови програмування мають справу із двома основними поняттями: даними й алгоритмами. Дані являють собою інформацію, що програма обробляє. А

алгоритми - це методи, які програма використає для обробки даних. Мова С, як і більшість основних мов програмування нашого часу, є процедурним. Це означає, що основна увага в ньому приділяється алгоритмам. Теоретично процедурне програмування полягає в тім, що спочатку визначається послідовність дій, що повинна бути виконана комп'ютером, а потім ці дії реалізуються за допомогою мови програмування. Програма містить набір процедур, які комп'ютер повинен виконати, щоб одержати необхідний результат. Це багато в чому схоже на те, як кулінарний рецепт пропонує послідовність дій (процедур), впливаючи яким кухар пече пиріг.

У міру того як програми ставали усе більше, перші процедурні мови, такі як FORTRAN й BASI, зіткнулися із проблемами організаційного плану. Наприклад, у програмах часто використовуються інструкції розгалуження, які направляють хід виконання програми убік того або іншого набору операторів залежно від результатів деякої перевірки. У багатьох старих програм такий заплутаний хід виконання (їх називають "програми-спагетти"), що зрозуміти їх, читаючи, власне кажучи, неможливо, а модифікація такої програми може привести до дійсної катастрофи. У відповідь на це вчені-"комп'ютерщики" розробили більше впорядкований стиль програмування, що називається структурне програмування.

Мова С включає ряд елементів, що полегшують застосування структурного програмування. Наприклад, структурне програмування обмежує можливості розгалуження (вибору наступного виконуваного оператора) невеликим набором добре функціонуючих конструкцій. Ці конструкції (цикли for, while, do while й оператор if else) входять у словник мови С.

Ще одним з нових принципів програмування було проектування програм зверху вниз. Ідея полягає в розбивці великої програми на менші, легше програмувальні завдання. Якщо одне із цих завдань як і раніше залишається занадто великої, розділіть її також на більше дрібні завдання. Продовжуйте цей процес доти, поки програма не буде розділена на маленькі, легко програмувальні модулі. (Наведіть порядок у своєму кабінеті. Ой! Добре,

наведіть порядок у письмовому столі, на столі й на своїх книжкових полках. Ох! Добре, почніть із письмового стола й наведіть порядок у кожному висувному ящику, починаючи із середнього. Гм, мабуть я можу впоратися із цим завданням.) Мова 3 полегшує такий підхід, заохочуючи програмістів розробляти програмні одиниці (елементи), називані функціями, які являють собою модулі окремих завдань. Як ви могли помітити, методика структурного програмування відбиває процедурний підхід, при якому програма розглядається з погляду виконуваних нею дій.

2.2 Об'єктно-орієнтоване програмування

Хоча принципи структурного програмування поліпшили зрозумілість і надійність програм, а також полегшили їхній супровід, створення програм більших розмірів як і раніше залишалося важким завданням.

Об'єктно-орієнтоване програмування (ООП) приносить із собою новий підхід до рішення цього завдання. На відміну від процедурного програмування, де головна увага приділяється алгоритмам, в ООП основну увагу спрямовано на дані. При використанні ООП проблему не вирішують за допомогою процедурного підходу, закладеного в мові, а пристосовують мову для рішення цієї проблеми. Ідея полягає в створенні таких форм даних, які відповідали б основним характерним рисам проблеми.

У мові C++ клас є специфікацією, що описує таку нову форму даних, а об'єкт - конкретною структурою даних, створеної відповідно до цієї специфікації. Наприклад, клас може описувати загальні властивості керівника корпорації (ім'я, посада, жалування й, наприклад, незвичайні здатності), тоді як об'єкт представляє конкретного керівника (Гилфорд Шипблат (Guilford Sheepblat), віце-президент, оклад \$325 тис. у рік, знає як користуватися файлом CONF1.SYS). Загалом, клас визначає, які дані будуть утворювати об'єкт й які операції можуть виконуватися над цими даними. Наприклад, припустимо, що ви розробляєте графічну програму, здатну малювати прямокутники. Можна

створити клас, що описує прямокутник. Даними в специфікації цього класу може бути наступне: місце розташування кутів, висота й ширина, колір і стиль обмежуючої лінії, а також колір і шаблон, використовувані для заповнення прямокутника. Частина специфікації цього класу, що описує операції, може включати методи переміщення прямокутника, зміни його розмірів, обертання трикутника, зміни квітів і шаблонів, а також копіювання прямокутника в інше місце. Якщо потім використати цю програму, щоб намалювати прямокутник, то вона створить об'єкт у відповідності зі специфікацією класу. Цей об'єкт буде містити всі значення даних, що описують прямокутник, а за допомогою методів класу можна буде цей прямокутник модифікувати. Якщо необхідно намалювати два прямокутники, то програма створить два об'єкти, по одному для кожного прямокутника.

Об'єктно-орієнтований підхід до розробки програми полягає в тому, що спочатку розробляються класи, що точно представляють ті речі, з якими має справу програма. У графічній програмі, наприклад, можна визначити класи для подання прямокутників, ліній, окружностей, кистей, ручок і т.п. Згадаєте, що визначення класу включає опис припустимих операцій для кожного класу, таких як переміщення окружності або обертання лінії. Після цього можна приступати до розробки самої програми, використовуючи об'єкти цих класів. Цей процес просування від більше низького рівня організації, такого як класи, до більше високого рівня, такому як програма, називається програмуванням знизу нагору.

Об'єктно-орієнтоване програмування - це не тільки зв'язування даних і методів у єдине ціле: визначення класу. ООП, наприклад, полегшує створення повторно використовуваного коду програми, що в остаточному підсумку звільняє від великого обсягу роботи. Приховання інформації дозволяє охоронити дані від небажаного доступу. Поліморфізм дає можливість створювати множинні визначення для операцій і функцій, а те, яке визначення буде використатися, залежить від контексту програми. Спадкування дозволяє створювати нові класи зі старих. Як бачите, з об'єктно-орієнтованим

програмуванням дозволяється багато нових ідей і використовується інший підхід до створення програм, чим при процедурному програмуванні. Замість того щоб зосередити свою увагу на завданнях, ви фокусуєте його на поданні різних понять. Замість того щоб використати програмування "униз", вам іноді доводиться використати програмування "нагору".

Одним з реальних переваг мови C++ є те, що він дозволяє легко адаптувати й повторно використати добре перевірені коди програм.

2.3 Узагальнене програмування

Узагальнене програмування - це ще одна парадигма програмування, підтримувана мовою C++. Воно має загальну з ООП мету - спростити повторне використання кодів програм і методів абстрагування загальних понять. Однак, у той час як в ООП основну увагу приділяється даним, в узагальненому програмуванні упор робиться на алгоритми. І в нього інша область застосування. ООП - це інструмент для розробки більших проєктів, тоді як узагальнене програмування надає інструменти для виконання завдань загального характеру, таких як сортування даних або об'єднання списків. Термін узагальнений означає створення коду програми, незалежного від типу даних. У мові C++ є дані різних типів - цілі числа, числа із дробовою частиною, символи, рядки символів, обумовлені користувачем складні структури, що складаються з даних декількох типів. Якщо, наприклад, потрібно сортувати дані різних типів, то звичайно для кожного типу створюється окрема функція сортування. Узагальнене програмування розширює мова таким чином, що дозволяє один раз написати функцію для узагальненого (тобто невизначеного) типу даних і потім використати неї для різноманітних реальних типів даних. Це забезпечується за допомогою шаблонів мови C++.

3 Мова C++ . Перспективи розвитку

Мова C++, так само як і мова C, є дітищем компанії Bell Laboratories. Як уже говорилося, Бьярні Страуструп розробив цю мову на початку 80-х років. По його власних словах, "мова C++ був спроектований головним чином так, щоб не доводилося програмувати на асемблері, C або різних сучасних мовах високого рівня. Його головна мета складалася в наступному: зробити так, щоб окремим програмістам було легше й приємніше писати гарні програми".

Страуструп був більше стурбований тим, щоб мова C++ був корисним, а не був носієм якої-небудь філософії або стилю програмування. Реальні потреби програмування більше важлива, чим теоретична чистота визначення властивостей мови. Страуструп створив C++ на основі мови C, тому що мова C був коротким, добре підходив для системного програмування, був широко доступний і тісно пов'язаний з операційною системою UNIX. Об'єктно-орієнтована частина мови C++ виникла під впливом мови моделювання Simula67. Страуструп додав елементи ООП у мову C, не змінюючи при цьому істотно сама мова C.

Таким чином, мова C++ є розширенням мови C, а це означає, що будь-яка правильна програма C є також правильною програмою C++. Є лише деякі незначні розходження, але вони не настільки істотні. Програми C++ можуть використати існуючі бібліотеки мови C. Бібліотеки - це сукупності програмних модулів, які можуть викликатися із програм. Вони надають готові рішення багатьох широко розповсюджених завдань програмування, заощаджуючи в такий спосіб багато часу й зусиль. Це допомогло поширенню мови C++.

Назва C++ походить від позначення оператора інкремента ++ у мові C, що додає одиницю до значення змінної. Назва C++ має на увазі, що ця мова є вдосконаленою (++) версією мови C.

Завдання, узяті з реального життя, комп'ютерна програма переводить у послідовність дій, що повинна бути виконана комп'ютером. Компонент ООП

у мові C++ дає можливість установлювати зв'язку між поняттями, що ставляться до даної проблеми, а елементи мови C забезпечують більше тісна взаємодія з апаратними засобами. Ця комбінація можливостей допомогла поширенню мови C++. Вона також може привести до необхідності змінювати методику програмування при переході від одного аспекту програми до іншого. (Справді , деякі прихильники чистого ООП вважають, що додавати елементи ООП до мови C - однаково , що прилаштовувати крила свині, нехай навіть продуктивної й симпатичної.) Крім того, оскільки C++ є мовою C, до якого додали елементи ООП, можна просто ігнорувати ці нові можливості. Але якщо ви так зробите, то багато чого втратите.

Тільки після того, як мова C++ одержав деяке визнання, Страуструп додав у нього шаблони. Елементи ООП забезпечують високий рівень абстрагування забезпечуючи тим самим можливість узагальненого програмування. І тільки після того, як шаблони були використані на практиці й удосконалені, він став розуміти, що вони мають таке ж значення, як й ООП, або навіть більше. Той факт, що мова C++ містить у собі як ООП, так й узагальнене програмування, показує, що в C++ упор робиться на утилітарний, а не ідеологічний підхід, і це одна із причин успіху цієї мови.

Питання для самоконтролю знань студентів

- 1 Які передумови створення мови програмування C?
- 2 Яка основна характеристика структурного (процедурного) програмування?
- 3 Дайте загальну характеристику об'єктно-орієнтованого програмування.
- 4 В чому полягає основна відмінність узагальненого програмування?
- 5 Які передумови створення мови C++?
- 6 Які перспективи розвитку мови програмування C/C++?

План самостійного вивчення теми № 3

з навчальної дисципліни Технології (Програмування)

Тема Операції над числовими типами даних: бінарні побітові операції

Мета Ознайомлення з операціями мови C/C++, що виконуються над двійковим представленням числових даних.

знати:

- форму запису та правила виконання побітових операцій.

вміти:

- обчислювати результат виконання побітових операцій.

Час – 4 години

Навчальні питання:

1 Операції зсуву

2 Порозрядні логічні операції

Методичні рекомендації

Самостійно вивчаючи запропоновану тему, студент повинен засвоїти такі основні положення:

По-перше Логічні операції, як й операції зрушення, працюють із послідовностями битов. Операція виконується окремо з кожною парою битов двох операндов.

По-друге Операції зсуву '>>' й '<<' зрушують біти цілочисельного значення вліво або вправо. Другий операнд задає число битів, на які зрушується перший операнд.

По-третє Порозрядні логічні операції містять у собі поразрядну операцію "І" (&), "виключаюче АБО" (^), "включаюче АБО" (|) і саму високопріоритетну операцію поразрядного доповнення - заперечення (~). Перші три операції двійкові, а остання - унарная (потрібно тільки один операнд).

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / 1О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Навчальні матеріали до самостійного вивчення теми

В С/С++ мається є 6 операцій для роботи з окремими бітами у внутрішньому двійковому уявленні числа. Їх можна здійснювати тільки над цілими операндами (char, int).

До таких операцій відносяться:

1 Операції зсуву

Формат виразу з операцією зсуву:

операнд_лівий операція_зсуву операнд_правий

Знаки операцій зсуву та правила виконання цих операцій представлені в таблиці 1.1

Таблиця 1.1 – Операції зсуву

Знак операції	Призначення
<<	Зсув вліво бітового уявлення значення лівого цілочисленого операнду на кількість розрядів, яке дорівнює кількості розрядів правого операнду. Зсув вліво є найбільш швидким способом множення числа на 2, 4, 8 і т.д. Приклад, n=13>>2 0000 0000 0000 1101 //13 15>>2 0000 0000 0000 0011 //2 n= 0000 0000 0000 0011 //3

Знак операції	Призначення						
>>	Зсув вправо бітового уявлення значення лівого цілочисленого операнду на кількість розрядів, яке дорівнює кількості розрядів правого операнду. Розряди, які звільняються обнулюються, якщо операнд беззнакового типу і заповнюються знаковим розрядом, якщо – знакового. Зсув вліво є найбільш швидким способом ділення числа на 2, 4, 8 і т.д. Приклад, n=5<<3 <table> <tr> <td></td><td>0000 0000 0000 0101 //5</td></tr> <tr> <td>5<<3</td><td>0000 0000 0010 1000 //40</td></tr> <tr> <td>n=</td><td>0000 0000 0010 1000 //40</td></tr> </table>		0000 0000 0000 0101 //5	5<<3	0000 0000 0010 1000 //40	n=	0000 0000 0010 1000 //40
	0000 0000 0000 0101 //5						
5<<3	0000 0000 0010 1000 //40						
n=	0000 0000 0010 1000 //40						

Операції '>>' й '<<' зрушують біти цілочисленого значення вліво або вправо. Другий операнд задає число битів, на які зрушується перший операнд. Насправді, всі не так погано, як звучить. Розглянемо спочатку операцію зсуву вправо:

```
int x=5, y=1, result; result = x >> y; // результат дорівнює 2
```

Дана операція зрушує послідовність битов лівого операнда (тут x, де втримується 5) на число позицій, які задає правий операнд (у цьому випадку у зі значенням 1). Двійкове подання 5 - це 101. Якщо зрушити дану послідовність битов на позицію вправо, вийде 10, що відповідає цілому значенню 2 (або ступінь двійки, задана другим операндом). Операція << зрушує біти у зворотному напрямку. Тут з 101 виходить 1010, що відповідає десятковому 10:

```
int x=5, y=1, result; result = x << y; // результат дорівнює 10
```

Коли біти зрушуються вліво, те перші біти операнда губляться, а праві біти заповнюються нулями (як в останньому прикладі). Аналогічно при зрушенні вправо губляться праві біти, але те, що відбувається з лівими бітами, залежить від машини. Самим лівим бітому цілого зі знаком (signed) буде біт знака. Якщо він дорівнює нулю, то число позитивне, а якщо 1 - негативне. Якщо число позитивне, то проблем немає: нульовий біт знака зрушується вправо, а нулі ліворуч займають його місце. Якщо ж значення негативне, то вправо зрушується одиниця, і виникає проблема переносимости. На деяких

машинах у біті знака виявляється одиниця (і поширюється далі). Це називається *арифметичним зрушенням*. На інші в біт знаке зрушується 0 (розповсюджуваний вправо). Таке зрушення називається *логічним*.

2 Порозрядні логічні операції

Знаки порозрядних логічних операцій та правила виконання цих операцій представлені в таблиці 1.2.

Таблиця 1.2 – Порозрядні логічні операції

Знак операції	Призначення
&	Порозрядна кон'юнкція (И) бітових уявлень значень цілих операндів (біт=1, якщо відповідні біти обох операндів =1). Часто використовується для обнулення деякої групи розрядів. Приклад, $n = 15 \& 3$ $n = 0000\ 0000\ 0000\ 1111\ //15$ $\& 0000\ 0000\ 0000\ 0011\ //3$ $n = 0000\ 0000\ 0000\ 0011$
	Порозрядна диз'юнкція (ИЛИ) бітових уявлень значень цілих операндів (біт=1, якщо відповідний біт хоча б одного із операндів =1). Часто використовується для встановлення 1 деякої групи розрядів. Приклад, $n = 3 16$ $n = 0000\ 0000\ 0000\ 0011\ //3$ $0000\ 0000\ 0001\ 0000\ //16$ $n = 0000\ 0000\ 0001\ 0011\ //19$
^	Порозрядна сума за модулем 2 (виключне ИЛИ) бітових уявлень значень цілих операндів (біт=1, якщо відповідний біт ТІЛЬКИ ОДНОГО з операндів =1). Приклад, $n = 5 \wedge 6$ $n = 0000\ 0000\ 0000\ 0101\ //5$ $\wedge 0000\ 0000\ 0000\ 0110\ //6$ $n = 0000\ 0000\ 0001\ 0011\ //3$
~	Порозрядне інвертування внутрішнього двійкового коду цілого операнду (побітове інвертування). Приклад, $n = \sim 65\ 535$

	n= 1111 1111 1111 1110 //65 534
	~
	n= 0000 0000 0000 0001 //1

Порозрядні логічні операції містять у собі поразрядну операцію "І" (&), "виключаюче АБО" (^), "включаюче АБО" (|) і саму високопріоритетну операцію поразрядного доповнення - заперечення (~). Перші три операції двійкові, а остання - унарная (потрібно тільки один операнд).

Логічні операції, як й операції зрушення, працюють із послідовностями битов. Операція виконується окремо з кожною парою битов двох операндов. Виконання цієї операції дає відповідний біт результату. Поразрядная операція "І" установлює біт результату в 1, якщо обоє відповідних біта операндов рівні 1, і в 0, якщо хоча б один із двох бітів дорівнює 0. У наступних прикладах розглядаються тільки чотири біти операнда й передбачається, що інші чотири біти містять нулі. Щоб проілюструвати операцію "І", припустимо, що перший операнд дорівнює 12 (1100 у двійковому поданні), а другий - 10 (1010 у двійковому виді). Зрівняємо кожен біт першого операнда із відповідним бітом другого. Як видно, тільки старший біт в обох операндах дорівнює 1. Інші містять різні значення (0 або 1). Отже, значенням результату буде 1000 (десятьокве 8): $1100 \& 1010$ дає 1000.

Поразрядная операція "включаюче АБО" установлює результат в 1, якщо один або обидва біти операнда рівні 1. Якщо обидва біти нульові, то результатом буде 0. Для операндів 12 (двійкове 1100) і 10 (двійкове 1010) всі біти результату, за винятком самого правого, установлюються в 1, що дає двійкове значення 1110 (десятьокве 14), тобто $1100 | 1010$ дає 1110.

Поразрядная операція "виключаюче АБО" установлює результат в 1, якщо тільки один з бітів операнда дорівнює 1. Якщо обидва біти містять однакове значення (0 або 1), то результатом буде 0. У нашому прикладі перший й останній біти операндов з, а друг і третій відрізняються, що дає двійковий код 0110 (десятьокве 6): $1100 \wedge 1010$ дає 0110.

Операція поразрядного доповнення встановлює біти результату у зворотні значення, тобто для одержання результату вона міняє біти операнда на протилежні. Якщо біт операнда дорівнює 1, то біт результату стає нульовим і навпаки. Наприклад, доповнення 12 (двійкове 1100) дає результат 0011 (десятькове 3), або ~ 1100 дає 0011.

Ці операції часто використовуються, коли в додатку обробляється великий обсяг інформації про стан. Наприклад, може бути включений або виключений комунікаційний канал, пристрій готовий або не готове, на провідник подається чи напруга ні, замовник заслуговує великої чи знижки ні, має гарний кредитний рейтинг і т.д. На великій машині можна дозволити собі виділити для кожного із цих значень ціле, хоча реально використовується тільки один біт (що містить 0 або 1). На малих комп'ютерах використовується набір булевих значень, для кожного з яких виділяється байт. От чому інформація такого роду часто впаковується в слова, так що кожен біт (прапор) у бітовій послідовності має окремий сенс. Щоб виділити зі слова стану значення конкретного біта або встановити його, використовуються операції зрушення й логічні операції з константами (конкретними послідовностями битов - масками).

Припустимо, третій біт праворуч у слові стану (нехай воно називається flags) означає, що пристрій включений. Коли пристрій включається, програма повинна встановлювати цей біт в 1.

Щоб установити біт в 1, потрібна змінна (назвемо її onMask), у якої третій біт має значення 1. Якщо застосувати до третього біта змінних flags й onMask операцію "включаюче АБО", то третій біт результату буде встановлюватися незалежно від того, який стан третього біта в змінній flags. Це швидше, ніж перевіряти даний біт, а потім виконувати операцію "включаюче АБО". Проблема в тім, що логічні операції не можна застосовувати до окремих биток - вони застосовуються одночасно до всіх биток операнда. Це означає, що всі біти змінної onMask (за винятком третього біта) повинні мати таке значення,

щоб інші біти змінної `flags` не мінялися. Для "включаючого АБО" біти повинні містити 0.

Саме так створюються маски для послідовностей битов: окремі біти встановлюються в значення, як того вимагає стан, а інші - у значення, не змінюючі існуючого стану. У даному прикладі в змінній `onMask` третій біт встановлюється в 1, а інші біти - в 0. Для чотирибітового подання одержуємо 0100 або десяткове 4:

```
int flags, onMask = 4;
flags = flags | onMask;    // встановлює третій біт в 1
```

Якщо пристрій виключений, це повинне відбиватися скиданням третього біта в 0 збереженням значень інших битов. Тоді буде потрібна інша маска (назвемо її `offMask`), що встановлює третій біт у 0, і логічна операція "І". Щоб інші біти не змінилися, потрібно встановити ці біти маски в 1. Отже, змінна `offMask` буде містити 1011 або десяткове 11.

Втім, тут є одна тонкість. Дана бітова послідовність дорівнює 11, тільки коли її розмір 4 біт. Для 8 біт послідовність буде мати вигляд 11111011, а десятковим значенням буде 244. Для 16 або 32 біт буде потрібно ще одне значення. От типовий приклад проблеми переносимості. Рішення тут досить прості. Всі ці бітові послідовності представляють заперечення бітового набору 0100 для слів різного розміру. Отже, стерпним методом ініціалізації змінної `offMask` буде використання бітового шаблону, зворотного 0100:

```
int OffMask = ~onMask;
flags = flags & offMask; // при цьому третій біт скидається в
0
```

Щоб перевірити, чи встановлений третій біт, можна застосувати операцію "І" до маски `onMask` і змінної `flags`. Ця операція встановлює всі біти результату в 0, за винятком третього біта (оскільки всі біти в масці `onMask`, крім третього, містять 0). Якщо третій біт змінної `flags` дорівнює 0 (пристрій виключений), то результатом буде 0 (false). Якщо третій біт змінної `flags`

дорівнює 0 (пристрій виключений), то результат відмінний від 0 (містить true). Ще один метод доступу до значень бітів складається в зрушенні послідовності битов прапора на дві позиції вправо й застосуванні операції "I" до прапора й маски, що містить всі нулі, крім правого біта. Якщо результатом буде 1, то біт установлений, а якщо результат нульовий, то біт також нульовий (про операції перевірки на рівність - див. нижче):

```
if (((flags >> 2)&1)==1) cout << "Третій біт установлений\n"; // перевірка
```

Ті, хто не збирається займатися розробкою програмного забезпечення для убудованих і комунікаційних систем, навряд чи стануть часто використати зрушення значень, а тому їм можна не занадто вникати в даний матеріал. Тим же програмістам, які планують подібну діяльність, краще попрактикуватися в застосуванні даних операцій. У таких системах вони дуже поширені.

Питання та завдання для самоконтролю знань студентів

- 1 Сформулюйте правила виконання наступних операцій:
 - Зсув вліво
 - Зсув вправо
 - Порозрядна кон'юнкція (И)
 - Порозрядна диз'юнкція (ИЛИ)
 - Порозрядна сума за модулем 2 (виключне ИЛИ)
 - Порозрядне інвертування
- 2 Як записується десяткове число 13 у двійковій системі числення?
- 3 Чому дорівнює значення змінної k після виконання команд:
`int n=3,m=2,k;`
`k=n&m;`
- 4 Чому дорівнює значення змінної k після виконання команд:
`int n=4,k=3;`
`k=n | k;`
- 5 Чому дорівнює значення змінної k після виконання команд:
`int n=6,k=10;`
`k=n^k;`
- 6 Чому дорівнює результат виконання операції:
`n= ~23 456`
- 7 Яка операція дозволяє швидким способом помножити число на 2, 4, 8 і т.д?
- 8 Яка операція дозволяє швидким способом поділити число на 2, 4, 8 і т.д?
- 9 Чому дорівнює результат виконання операції:
`24>>3`
- 10 Чому дорівнює результат виконання операції:
`12<<2`

План самостійного вивчення теми № 4

з навчальної дисципліни Технології (Програмування)

Тема Стандартні функції обробки рядків символів.

Мета Охарактеризувати та навчити студентів використовувати стандартні функції обробки рядків символів

знати:

- основні функції обробки рядків символів та функції перетворення.

вміти:

- використовувати функції обробки рядків символів для розв'язання прикладних задач.

Час – 6 годин

Навчальні питання:

1 Основні функції обробки рядків символів

2 Функції перетворення

Методичні рекомендації

Самостійно вивчаючи запропоновану тему, студент повинен засвоїти такі основні положення:

По-перше Для роботи з рядками символів існують спеціальні бібліотечні функції, які містяться в заголовних файлах `string`, `stdlib.h`,

По-друге Основні функції обробки рядків символів використовуються для:

- обчислення довжини рядків;
- конкатенації (склеювання) рядків;
- порівняння рядків;
- пошуку підрядків;
- копіювання рядків
- пошуку входження символів в рядки;
- модифікації рядків.

По-третє Функції перетворення, що здійснюють зв'язок символьного типу з числовими типами, як правило

використовуються для перевірки коректності введених користувачем даних.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Навчальні матеріали до самостійного вивчення теми

1 Основні функції обробки рядків символів

Для роботи з рядками символів існують спеціальні бібліотечні функції, які містяться в заголовному файлі `string.h`. Розглянемо функції стандартної бібліотеки для роботи з рядками на прикладі. Нехай

```
char *s1="Программируем на \n";  
char *s2="языке C/C++\n", s3;  
int k, n;
```

Заголовний файл `<string>`

1 Опис функції: `char *strcat (char *s1, char *s2);`

Призначення: Додає `s2` до `s1` і результат повертає в `s1`, додаючи в кінець нуль-символ. Нульовий символ рядка `s1` перекривається першим символом рядка `s2`. **Розмір рядка `s1` повинен бути достатнім для зберігання як `s1`, так і `s2`.**

Команда: `strcat(s1,s2)`

Результат: `s1="Програмуємо на мові C/C++\n"`

2 Опис функції: `char *strchr(char *s, int ch);`

Призначення: Повертає підрядок `s1`, починаючи з першого входження символу `ch` в рядок `s1`. Якщо входження не виявлено – повертає `0`.

Команда: `s3=strchr(s1,'м');`

Результат: `s3="ммуємо на \n"`

3 Опис функції: `int strcmp(char *s1, char *s2);`

Призначення: Порівнює рядки. Посимвольно порівнюються коди символів до першої пари нерівних символів. Повертає ціле, негативне - якщо `s1 < s2`, нуль – якщо `s1 = s2` та ціле позитивне - якщо `s1 > s2`.

Команда: `k=strcmp(s1,s2);`

Результат: `k=-41` тому, що код символу 'П' менше коду символу 'я'

4 Опис функції: `char *strcpy(char *s1, char *s2);`

Призначення: Копіює `s2` в `s1` і повертає `s1`. Якщо довжина `s1` менше довжини `s2`, результат буде невизначеним.

Команда: `strcpy(s1,s2);`

Результат: `s1="мові C/C++\n"`

5 Опис функції: `int strcspn(char *s1, char *s2);`

Призначення: Повертає індекс першого символу в рядку s1, що міститься в s2. Інакше, повертає довжину початкового підрядка s1, що не містить символів з рядка s2.

Команда: k= strchr(s1, s2);

Результат: k=13 тому, що 13-й символ рядка s1 (починаючи з нульового) - символ пробілу, це перший символ, який міститься в рядку s2.

6 Опис функції: int strlen(char *s);

Призначення: Повертає довжину рядка (без урахування символу завершення рядка)

Команда: k= strlen(s1);

Результат: k=17

7 Опис функції: char *strncat(char *s1, char *s2, int n);

Призначення: Складає рядок s1 з n символами рядка s2. *Розмір рядка s1 повинен бути достатнім для зберігання як s1, так і n символів s2.*

Команда: strncat(s1, s2, 5);

Результат: s1="Программируем на языке \n"

8 Опис функції: int *strncmp(char *s1, char *s2, int n);

Призначення: Порівнює перші n символів рядка s1 з n першими символами рядка s2.

Команда:

Результат:

9 Опис функції: char *strncpy(char *s1, char *s2, int n);

Призначення: Копіює перші n символів рядка s2 в рядок s1. *Довжина s1 повинна бути більше n, інакше – результат невизначений.*

Команда: `strncpy(s1, s2, 4);`

Результат: `s1="языкраммируем на \n"` *Поясніть отриманий результат!*

10 Опис функції: `int strcspn(char *s1, char *s2);`

Призначення: Повертає індекс першого символу в `s1`, що не міститься в `s2`. Інакше, повертає довжину початкової підстроки `s1`, що містить тільки символи із `s2`.

Команда: `k= strspn(s1, s2);`

Результат: `k=0` тому, що 0-й символ рядка `s1` - символ 'П', це перший символ, який не міститься в рядку `s2`.

11 Опис функції: `char *strstr(char *s1, char *s2);`

Призначення: Повертає підстроку `s1`, починаючи з якої `s2` повністю входить в `s1`. Якщо `s2` не є підстрокою `s1` – результат невизначений.

Команда: `s3=strstr(s1, s2);`

Результат: Результат не визначений.
Якщо `s2="мируем\n"`, то отримали б результат:
`s3="мируем на \n"`,

12 Опис функції: `char *strlwr(char * s);`

Призначення: Перетворить прописні символи рядка в рядкові.

Команда: `strlwr(s1);`

Результат: Якщо `s1="Programer\n"`, то отримали б результат:
`s1="programer\n"`.

13 Опис функції: `char *strupr(char * s);`

Призначення: Перетворить рядкові символи рядка в прописні.

Команда: `strupr(s1);`

Результат: Якщо `s1="Programer\n"`, то отримали б результат:
`s1="PROGRAMER\n"`.

14 Опис функції: `char *strset(char * s, char c);`

Призначення: Заповнює рядок зазначеним при виклику функції символом.

Команда: `strset(s1, 'a');`

Результат: `s1="aaaaaaaaaaaaaaaa\n"`;

2 Функції перетворення

Заголовний файл `<stdlib.h>`

1 Опис функції: `double atof(const char *str);`

Призначення: Перетворює рядок `str` в значення типу `double`. Рядок повинен містити коректне представлення числа з плаваючою крапкою. Інакше функція поверне невизначене (тобто `NULL`) значення.
Число у вхідному потоці може завершуватись будь яким символом, недопустимим для представлення дійсних чисел: роздільником, знаком пунктуації (але не крапкою), буквеним символом, відмінним від букв `E` та `e`. Це означає, що при виклику `atof("13.23 Hello")` функція поверне число 13.23.

2 Опис функції: `int atoi(const char *str);`

Призначення: Перетворює рядок `str` в значення типу `int`. Рядок повинен містити коректне представлення цілого числа. Інакше функція поверне невизначене (тобто `NULL`) значення.
Число у вхідному потоці може завершуватись будь яким символом, недопустимим для представлення цілих чисел: роздільником, знаком пунктуації, буквеним символом. Це означає, що при виклику

atoi("123.23") функція поверне число 123, а дробова частина ".23" буде проігнорована.

3 Опис функції: long atol(const char *str);

Призначення: Перетворює рядок str в значення типу long. Рядок повинен містити коректне представлення довгого цілого числа. Інакше функція поверне невизначене (тобто NULL) значення.
Число у вхідному потоці може завершуватись будь яким символом, недопустимим для представлення цілих чисел: роздільником, знаком пунктуації, буквеним символом. Це означає, що при виклику atol("123.23") функція поверне число 123, а дробова частина ".23" буде проігнорована.

4 Опис функції: double strtod(const char* str, char **end);

Призначення: Перетворює рядок str в значення типу double. Функція працює наступним чином. Спочатку із рядка str виключаються всі роздільники. Потім зчитуються символи, що утворюють число. Якщо будь який із зчитаних символів не може бути елементом запису числа, процес припиняється і функція повертає значення 0. Інакше, вказівник end встановлюється на залишок рядка str, якщо він існує. Це означає, що strtod("12.34 Hello", end) поверне число 12.34, а вказівник end буде встановлено на пробіл перед Hello.

В якості прикладу використання наведених функцій розглянемо програму, яка блокує некоректне введення користувачем числа типу float.

Текст програми:

```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
int main ()
{
    int k=0;    // прапорець припинення некоректного введення
    float x;
```

```

cout<<endl<<" Введите действительное число"<<endl;
do
{
    gets(s3);      // зчитування рядка з вхідного потоку
    cin.sync();
    if (strtod(s3,end)==0)
    {
        cout<<endl<<"Некорректный ввод действительного числа. "<<endl;
        cout<<endl<<"Повторите ввод. "<<endl;
    }
    else
    {
        x=strtod(s3,end);
        if (strlen(*end)!=0)
        {
            cout<<endl<<"Некорректный ввод действительного числа. "<<endl;
            cout<<endl<<"Повторите ввод. "<<endl;
        }
        else
        { k=1;
          cout<<endl<<"Вы ввели действительное число"<<x<<endl;
        }
    }
} while (k==0);
getch();
return 0;
}

```

2 Обробка масивів рядків символів

Для обробки масиву рядків символів доцільно використовувати динамічний двовимірний масив. При цьому кількість рядків в масиві може задавати користувач:

```

...
int str;      // кількість рядків в масиві
cout<<"Ведите количество строк"<<endl;
cin>>str;

// створення вказівника на вказівник char
// і виділення пам'яті під str вказівників на str рядків
char **ms=new char *[str];

```

звільнення пам'яті в динамічній області відбуватиметься за допомогою інструкції

```
delete []ms;
```

Приклад програми обробки масиву рядків.

Постановка задачі.

Для введеного користувачем масиву рядків символів вивести найдовший рядок (або рядки, якщо їх декілька). Кількість рядків в масиві задає користувач.

Текст програми

```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
void main ()
{
    int i;
    int str; // кількість рядків
    int max; //для знаходження максимальної довжини рядка
    cout<<"Введіть кількість рядків в масиві"<<endl;
    cin>>str;
    // виділення пам'яті під масив рядків
    char **ms=new char *[str];
    //виділення пам'яті масив довжин рядків
    //i-й елемент масиву len дорівнюватиме довжині i-го рядка
    int *len=new int [str];
    // цикл введення рядків символів
    // та одночасне заповнення масиву len
    for (i=0; i<str; i++)
    { gets(*(ms+i));
      *(len+i)=strlen(*(ms+i));
      cout<<*(len+i)<<endl;
    }
    //пошук максимального елементу масиву len
    //тобто пошук максимальної довжини рядка в масиві
    max=*len;
    for (i=1; i<str; i++)
        if (max<*(len+i)) max=*(len+i);

    // цикл виведення рядків масиву з максимальною довжиною
    for (i=0; i<str; i++)
        if (*(len+i)==max) puts(*(ms+i));
    // звільнення динамічної пам'яті
    delete [] ms;
    delete [] len;
    getch();
}
```

```
return;  
}
```

Питання та завдання для самоконтролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу

- 1 Яка функція використовується для визначення довжини рядка?
- 2 Яка функція використовується для копіювання рядків?
- 3 Яка функція використовується для порівняння двох рядків?
- 4 Яка функція використовується для "склеювання" двох рядків?
- 5 Яке призначення функцій перетворювання?
- 6 Як працює функція `atoi()`?
- 7 Як працює функція `atof()`?
- 8 Яке призначення функції `strtod()`?
- 9 Як доцільніше розташувати в пам'яті масив рядків символів? Як при цьому здійснити доступ до *j*-го символу *i*-го рядка?

Завдання для самоперевірки засвоєного матеріалу

- 1 Розробити програму блокування некоректне введення користувачем числа типу `int`.
- 2 Розробити програму блокування некоректне введення користувачем числа типу `double`.
- 3 Для введеного користувачем масиву із *m* рядків символів (*m* вводять користувач) розробити програму пошуку та виведення тих рядків, які містять найбільше слів (підпоследовностей символів, що не містять пробілу всередині себе).
- 4 Для введеного користувачем масиву із *n* рядків символів (*n* вводять користувач) розробити програму, яка перше слово кожного рядка символів переписує в зворотному порядку. Вивести оновлений масив рядків.

План самостійного вивчення теми № 5

з навчальної дисципліни Програмування

Тема Обробка статичних масивів даних з використанням
вказівників

Мета Навчити використовувати вказівники для звернення до елементів
лінійних масивів та матриць.

знати: правила використання покажчиків для доступу до елементів лінійного
масиву та матриці.

вміти: використовувати покажчики для доступу до елементів лінійного масиву
та матриці.

Час – 4 години

Навчальні питання:

1 Приклади програм обробки статичних масивів з використанням
вказівників.

1.1 Приклади програми обробки лінійного масиву з використанням
вказівників.

1.2 Приклад програми обробки матриці з використанням
вказівників.

Методичні рекомендації

Самостійно вивчаючи запропоновану тему, студент повинен засвоїти
такі основні положення:

По-перше Для доступу до елементів лінійного масиву ми
використовували операцію індексування
ім'я_масиву[індекс] – це вираз з двома операндами, де
ім'я_масиву – це вказівник константа, а індекс визначає зсув
від початку масиву. Використовуючи вказівники, звернення
по індексу можна записати таким чином:
*(ім'я_масиву+індекс);

По-друге

Оскільки матриця (двомірний масив) являє собою список одноірних масивів, то

Використовуючи вказівники, звернення по індексам до елементів матриці можна записати таким чином:

`*(им'я_масиву+індекс_1)+індекс_2);`

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Навчальні матеріали до самостійного вивчення теми

1 Приклади програм обробки статичних масивів з використанням вказівників

1.1 Приклади програми обробки лінійного масиву з використанням вказівників.

Для доступу до елементів лінійного масиву ми використовували операцію індексування `ім'я_масиву[індекс]` – це вираз з двома операндами, де `ім'я_масиву` – це вказівник константа, а `індекс` визначає зсув від початку масиву. Використовуючи вказівники, звернення по індексу можна записати таким чином: `*(ім'я_масиву+індекс);`

Приклад 1.

Постановка задачі

Дано масив цілих чисел `M(11)`. Помістити в новий масив тільки від'ємні непарні елементи вхідного масиву.

Блок – схема алгоритму рішення задачі наведена на рисунку 1.1:

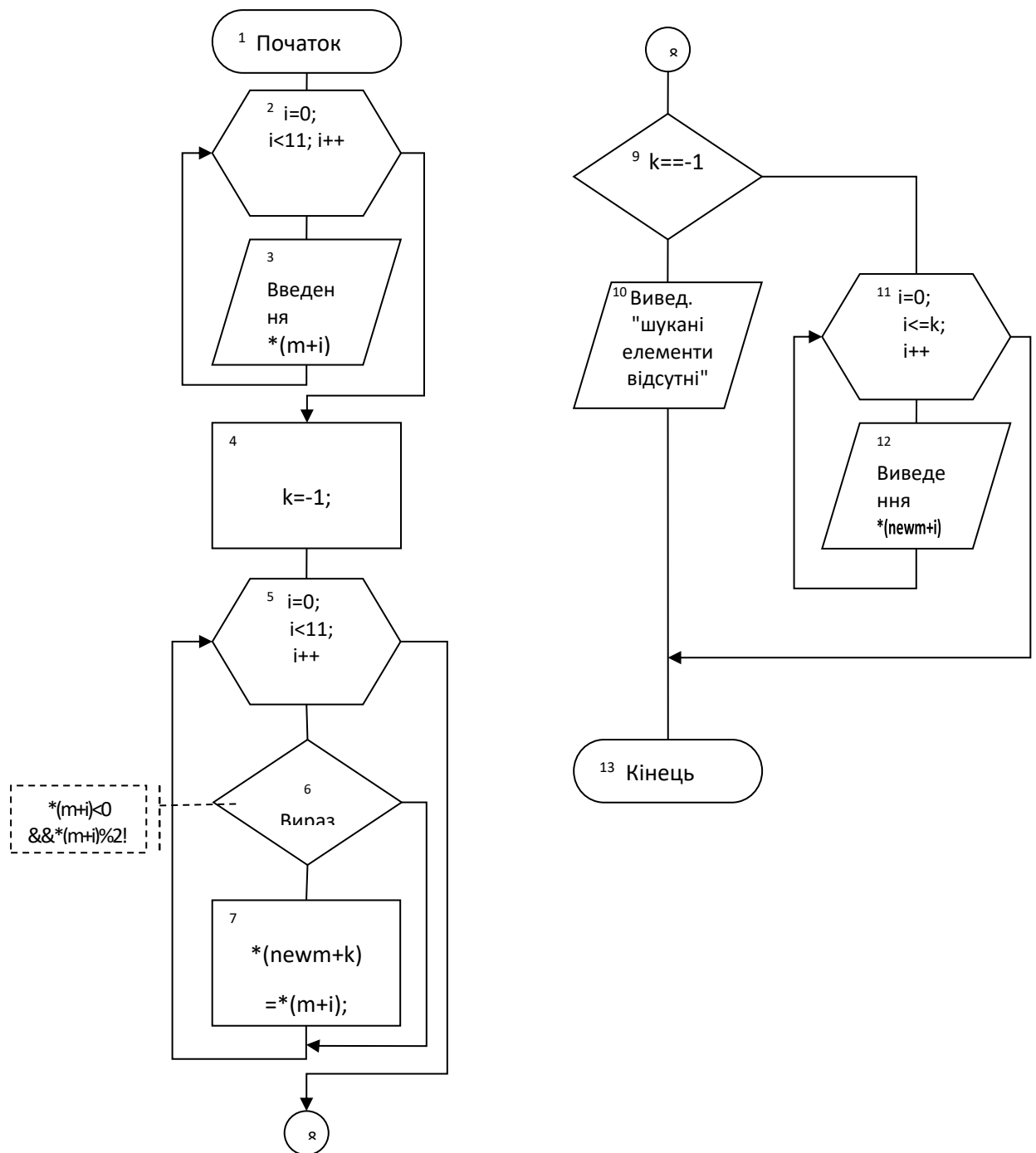


Рисунок 1.1. Блок-схема алгоритму рішення задачі

Текст програми:

```

#include <iostream.h>
#include <conio.h>
int main()
{

```

```

int m[11]; //опис вхідного масиву
int newm[11]; //опис вихідного масиву
int k; //фактична кількість елементів вихідного масиву
int i;
clrscr();
// Ведення елементів масиву
for( i=0;i<11;i++)
{
    cout<<endl<<"Введіть M["<<i+1<<"]:";
    cin>>*(m+i);
}
// пошук від'ємних непарних у вхідному масиві, та розміщення їх в
вихідному масиві
k=-1;
for(i=0;i<11;i++)
    if (*(m+i)<0 && *(m+i)%2!=0)
    {
        k++;
        *(newm+k)=*(m+i);
    }
/*якщо у вхідному масиві не має від'ємних непарних елементів,
k так і залишиться рівним -1 */
if (k== -1)
    cout<<endl<<"Непарні від'ємні елементи в масиві відсутні";
else
{
    // Виведення нового масиву
    clrscr();
    cout<<endl<<"Новий масив:"<<endl;
    for(i=0;i<=k;i++)
        cout<<*(newm+i)<<"    ";
}
cout<<endl<<"Для завершення програми натисніть Enter ";
getch();
return 0;
}

```

Приклад 2.

Постановка задачі.

Дано масив дійсних чисел M(11). Використовуючи вказівники, розробити програму пошуку мінімального серед тих елементів масиву, значення яких належать діапазону [-5;5].

Текст програми.

```

#include <iostream.h>
#include <conio.h>
int main()
{
    float m[11];
    float *pm=m;
    float min=6;
    clrscr();
    //введення елементів масиву
    for(int i=0;i<11;i++)
    {
        cout<<endl<<"Введіть M["<<i+1<<"]:";
        cin>>*(pm+i);
    }
    //виведення масиву на екран
    clrscr();
    cout<<endl<<"Введений масив:"<<endl;
    for(int i=0;i<11;i++) cout<<*(pm+i)<<"  ";
    //пошук мінімального елемента з діапазону
    for(int i=0;i<11;i++)
        if (*(pm+i)>=-5 && *(pm+i)<=5 && min>*(pm+i))
            min=*(pm+i);
    if (min!=6)
        cout<<endl<<"Мінімальний елемент з діапазону [-5;5]="<<min;
    else
        cout<<endl<<"Елемент з діапазону [-5;5] - відсутні";
    getch();
    return 0;
}

```

1.2 Приклад програми обробки матриці з використанням вказівників.

Оскільки матриця (двомірний масив) являє собою список одновірних масивів, то для того щоб оголосити двомірний масив необхідно задати дві розмірності.

тип ім'я_масиву[індекс_1][індекс_2];

де тип – визначає тип даних елементів, з яких буде складатися масив,

ім'я_масиву – ідентифікатор масиву,

індекс_1 – кількість рядків матриці,

індекс_2 – кількість стовпців матриці.

Для доступу до елементів матриці ми використовували операцію індексування `ім'я_масиву[індекс_1][індекс_2]` – це вираз з трьома операндами, де `ім'я_масиву` – це вказівник константа, `індекс_1` визначає зсув від початку масиву на `індекс_1` рядків (тобто блоків розміром в кількість стовпців в матриці), а `індекс_2` визначає зсув від початку відповідного блоку до шуканого елемента. Використовуючи вказівники, звернення по індексам до елементів матриці можна записати таким чином: `*(ім'я_масиву+індекс_1)+індекс_2`;

Постановка задачі.

Дано матрицю дійсних чисел, що містить 5 рядків та 7 стовбців. Використовуючи вказівники, розробити програму пошуку суми додатних елементів в кожному стовбці матриці.

Текст програми.

```
#include <iostream.h>
#include <conio.h>
int main()
{
    float m[5][7]; // опис матриці
    float s; // змінна для підрахунку суми додатних в кожному стовбці
    int i, j;
    clrscr();
    // введення елементів матриці
    for (i=0; i<5; i++)
        for (j=0; j<7; j++)
        {
            cout<<endl<<"Введіть M["<<i+1<<"]["<<j+1<<"]: ";
            cin>>*(m+i);
        }
    // виведення елементів матриці
    cout<<endl<<"Введённая матрица "<<endl;
    for (i=0; i<5; i++)
    {
        for (j=0; j<7; j++)
        {
            cout.width(3); /* встановлення ширини виводу 3 позиції */
            cout<<* (m+i)+j);
        }
        cout<<endl;
    }
    // пошук суми додатних
    for (j=0; j<7; j++)
```

```

{
    s=0;
    for(i=0;i<5;i++)
        if (*(m+i)+j)>=0)
            s=s+*(m+i)+j;
    if (s!=0)
        cout<<endl<<"сумма положительных элементов в столбце "<<j+1<<"
равна "<<s;
    else
        cout<<endl<<"положительных элементов столбце "<<j+1<<"
нет ";
}
getch();
return 0;
}

```

Питання та завдання для самоконтролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу

- 1 Чи вірне твердження ім'я_масиву=&ім'я_масиву[0]=&ім'я_масиву?
- 2 Чи вірне твердження ім'я_масиву=&ім'я_масиву[1]?
- 3 Чи вірне твердження *(a++), якщо a – ім'я масиву?
- 4 Чи вірне твердження *(a+1), якщо a – ім'я масиву?
- 5 Яка загальна форма звернення до елементу лінійного масиву за допомогою вказівника?
- 6 Яка загальна форма звернення до елементу матриці за допомогою вказівника

Завдання для самоперевірки засвоєного матеріалу

Даний масив MAS(10). Обробити його за наступним алгоритмом, використовуючи для організації доступу до елементів масиву вказівники.

Виводити результати роботи програми за першою та другою частинами завдання послідовно.

1. Знайти мінімум парних елементів масиву та його порядковий номер(номери). Відсортувати масив за незростанням елементів, виключаючи з масиву додатні елементи.

2. Знайти мінімум додатних елементів масиву та його порядковий номер(номери). Відсортувати масив за незростанням елементів, виключаючи з масиву від'ємні елементи.

3. Знайти мінімум від'ємних елементів масиву та його порядковий номер(номери). Відсортувати масив за незменшенням елементів, виключаючи з масиву додатні елементи.

4. Знайти мінімум непарних елементів масиву та його порядковий номер(номери). Відсортувати масив за незростанням елементів, виключаючи з масиву парні елементи.

5. Знайти мінімум з діапазону $[-6 ; 4]$ Відсортувати масив за незменшенням елементів, виключаючи з масиву парні елементи..

План самостійного вивчення теми № 6

з навчальної дисципліни Технології (Програмування)

Тема Вказівники на структури. Передача структур функціям.

Мета Оволодіння студентами правил передачі структури як параметра функції.

знати:

- правила опису вказівника на структуру;
- правила доступу до полів структури через вказівник;
- форми передачі структури за значенням, по посиланню та за адресою;

вміти:

- використовувати структури як параметри функцій;
- характеризувати компоненти мови програмування.

Час – 2 години

Навчальні питання:

- 1 Вказівники на структури
- 2 Структури як параметри функцій
- 3 Сортування по ключу

Методичні рекомендації

Самостійно вивчаючи запропоновану тему, студент повинен засвоїти такі основні положення:

По-перше Вказівники на структури визначаються також як і вказівники на інші типи даних.

По-друге Вказівник на структуру забезпечує доступ до її елементів 2 способами:

- 1 (*покажчик).ім'я_поля
- 2 вказівник->ім'я_поля

По-третє Структури, так само, як і будь-які інші типи, можуть бути параметрами функцій. Існують три способи передачі

структур у функції: передача за значенням, передача по посиланню та передача за адресою.

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Навчальні матеріали до самостійного вивчення теми

1 Вказівники на структури

Вказівники на структури визначаються також як і вказівники на інші типи даних.

Загальний вигляд описання вказівника на структуру.

```
ім'я_типу * ім'я_вказівника;
```

Приклад.

```
struct Student  
{char name[30];  
char group[10];  
float rating;  
};  
Student *ps;
```

Можна ввести вказівник для типу struct, що не має імені (спосіб визначення 2):

```
struct  
{  
char *name;
```

```
int age;  
} *person;      //вказівник на структуру
```

При визначенні вказівник на структуру може бути відразу ж проініціалізований.

```
Student *ps=&mas[0];
```

Вказівник на структуру забезпечує доступ до її елементів 2 способами:

- 1 (*показчик) .ім'я_поля
- 2 вказівник->ім'я_поля

Приклад.

```
cin>>(*ps).name;  
cin>>ps->rating;
```

2 Структури як параметри функцій

Структури, так само, як і будь-які інші типи, можуть бути параметрами функцій. Покажемо три способи передачі структур у функції й опишемо їх відмінності.

Нехай в програмі описано структуру, що містить інформацію про книгу в каталозі бібліотеки (автора, назву книги, рік видання, кількість сторінок).

```
struct Book  
{  
    char author[40];      // автор, символьний рядок  
    char title[80];       // назва, символьний рядок  
    int year;             // рік видання, ціле число  
    int pages;            // кількість сторінок, ціле число  
};
```

Розглянемо функцію, що записує в поле year (рік видання книги) значення 2002.

Передача за значенням

Якщо параметр функції оголошений як:

```
void Year2002( Book b )
{
    b.year = 2002;
}

void main()
{
    Book b;
    Year2002 ( b );
}
```

то при роботі функції створюється копія цієї структури в стеці (спеціальної області пам'яті, де зберігаються параметри й локальні змінні процедур і функцій) і функція працює із цією копією. Такий підхід має два недоліки:

По-перше, функція не змінює поля структури в зухвалій програмі. Тобто , у нашому випадку завдання не вирішується.

По-друге, саме головне - структури можуть бути досить великих розмірів і створення нової копії може вичерпати стек (за замовчуванням він усього 4 Кб).

Передача по посиланню

Тому найчастіше параметри-структури передаються в функції по посиланню (при оголошенні за ім'ям типу структури ставлять знак &).

```
void Year2002( Book &b )
{
    b.year = 2002;
}
```

У цьому випадку фактично в функцію передається адреса структури й функція працює з тим же екземпляром, що й основна програма. При цьому всі зміни, зроблені в функції, залишаються в силі. Робота зі структурою-параметром усередині функції й виклик цієї функції з основної програми ніяк не відрізняються від попереднього варіанта.

Передача за адресою

Передача по посиланню можливе тільки при використанні мови C++ (вона не входить у стандарт мови C). Проте, у класичному C можна передати як параметр адресу структури. При цьому звертатися до полів структури усередині процедури треба через оператор `->`.

```
void Year2002( Book *b )
{
    b->year = 2002;
}

void main()
{
    Book b;
    Year2002 ( &b );
}
```

параметр - адреса структури

звертання до поля структури

передається адреса структури

3 Сорткування по ключу

Оскільки в структури входить багато полів, виникає питання: а як їх сортувати? Це залежить від конкретного завдання, але існують загальні принципи, про які ми й будемо говорити.

Для сортування вибирається одне з полів структури, воно називається ключовим полем або просто ключем. Структури розставляються в певному порядку за значенням ключового поля, уміст всіх інших полів ігнорується.

І ще одна проблема - при сортуванні елементи масиву міняються місцями. Структури можуть бути досить більших розмірів, і копіювання цих структур буде займати дуже багато часу. Тому сортування масиву структур виконують по покажчиках.

Ми вже використали цей прийом для сортування масиву символьних рядків. У пам'яті формується масив покажчиків `p`, і спочатку вони вказують на структури в порядку розташування їх у масиві, тобто покажчик `p[i]` вказує на структуру з номером `i`. Потім залишається тільки розставити покажчики так, щоб ключові поля відповідних структур були відсортовані в заданому порядку. Для рішення завдання зручно оголосити новий тип даних `PBook` – покажчик на структуру `Book`.

```
typedef Book * PBook;
```

Приклад нижче показує сортування масиву структур по році випуску книг. Основний алгоритм сортування масиву покажчиків закладений у функцію `SortYear`, що має два параметри - масив покажчиків і число структур.

```
void SortYear ( PBook p[], int n )
{
    int i, j;
    PBook temp;
    for ( i = 0; i < n-1; i ++ )
        for ( j = n-2; j >= i; j -- )
            if ( p[j+1]->year < p[j]->year )
            {
                temp = p[j];
                p[j] = p[j+1];
                p[j+1] = temp;
            }
}
```

тимчасовий покажчик

«бульбашковий» метод

порівняння ключових

полів

якщо покажчики розміщені
неправильно,
міняємо їх місцями

```
}
```

Основна програма, що виводить на екран інформацію про книги в порядку їхнього виходу, виглядає приблизно так:

```
#include <stdio.h>
const N = 20;
struct Book {
char author[40];
char title[80];
int year;
int pages;
};
typedef Book *PBook;
... // тут треба розташувати процедуру SortYear
void main()
{
Book B[N];
PBook p[N];
// тут введення даних у структури
for ( i = 0; i < N; i ++ )
p[i] = &B[i];
SortYear ( p, N );
for ( i = 0; i < N; i ++ )
printf("%d: %s\n", p[i]->year, p[i]->title);
```

сконструйований тип даних

масив покажчиків

розставити покажчики

сортування по покажчиках

виведення через покажчики

Питання та завдання для самоконтролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу

- 1 Загальний вигляд описання вказівника на структуру.
- 2 Показати 2 способи доступу до елементів структури через вказівники.
- 3 Описати три способи передачі структур у функції
 - передача за значенням;
 - передача по посиланню;
 - передача за адресою;

Завдання для самоперевірки засвоєного матеріалу

Дано масив, що містить наступну інформацію о студентах коледжу:

- прізвище, ім'я, по-батькові;
- номер групи;
- оцінки по п'яти екзаменам;
- середній бал;
- коефіцієнт.

Використовуючи передачу структур як параметрів, написати функції для розв'язку наступних задач:

- 1 Введення даних про студентів. Кількість студентів задає користувач.
- 2 Обчислення середнього балу студентів
- 3 Нарахування коефіцієнту за правилом: відмінники – 1,5; студенти, що мають лише одну оцінку «4» - 1,2; студенти с середнім балом від «4» до «4,5» - 1; студенти, що мають лише одну оцінку «3» - 0,8; студенти с середнім балом нижче «3,8» - 0.

План самостійного вивчення теми № 7

з навчальної дисципліни Технології (Програмування)

Тема: Шаблон `auto_ptr`. Додаткові функції стандартного введення – виведення.

Мета: Ознайомитись з синтаксисом та принципами роботи та використання функцій: `intungetc(int c, FILE *fp)`, `intfflush(FILE *fp)`, `intsetvbuf(FILE *fp, char *buf, intmode, sizejsize)`, `fread` і `fwrite`, `size_tfwrite(void *ptr, sizejsize, sizejnmemb, FILE *fp)`

знати: Студент повинен знати синтаксис (форму запису) функцій стандартного введення – виведення при роботі з файлами.

вміти: Студенти повинні навчитись правильно використовувати функції стандартного введення – виведення.

Час – 4 години

Навчальні питання:

- 1 Функція `intungetc(int c, FILE *fp)`
- 2 Функція `intfflush(FILE *fp)`
- 3 Функція `intsetvbuf(FILE *fp, char *buf, intmode, sizejsize)`
- 4 Функції `fread` і `fwrite`
- 5 Функція `size_tfwrite(void *ptr, sizejsize, sizejnmemb, FILE *fp)`

Методичні рекомендації

Самостійно вивчаючи запропоновану тему, студент повинен засвоїти такі основні положення:

По-перше, в C++, коли програма зчитує з файлу останній елемент, умова кінця файлу не виникає. Воно виникає при наступному читанні, коли програма намагається вважати дані за останнім елементом у файлі. Уникайте подібної ситуації.

По-друге, після збереження на дані не вплине аварійне або нормальне завершення програми або збій харчування. Крім того, що ще більш важливо,

дані можуть використатися іншою програмою іншим часом й/або в іншому місці. Це істотно розширює гнучкість інформаційних систем.

По-третьє, в програмі, що виконує операції читання з файлу або запис у файл, повинна бути оголошена змінна-показчик на тип FILE;

По-четверте, при роботі з файлами можливі помилки, тому рекомендується за допомогою функції `ferror` перевіряти результат виконання потенційно небезпечних, з погляду виникнення помилок, операцій з файлами (`fopen`);

ЛІТЕРАТУРА

- 1 О.Г. Трофименко. С++. Основи програмування. Теорія та практика: [Текст] / О.Г. Трофименко. – Одеса: "Фенікс", 2010. – 544 с.: іл.
- 2 Браян В. Керніган, Деніс М. Річі. . Мова програмування С. (друге видання) [Електронний ресурс] /<http://m.programming.in.ua/programming/c-language/227-book-programming-c-kernighan.html>

Навчальні матеріали до самостійного вивчення теми

1 Функція `intungetc(int c, FILE *fp)`

Функція `intungetc(int c, FILE *fp)` поміщає символ, визначений перемінною `c`, назад у вхідний потік.. Після цього при наступному виклику стандартної функції введення цей символ буде лічений. Представите, наприклад, що потрібно функція, що зчитує всі символи, крім двокрапки. Можна використовувати `getchar` чи `getc()` для читання символів доти, поки не потрапиться двокрапка і потім викликати `ungetc()`, щоб помістити його назад у вхідний потік. Стандарт ANSI 3 гарантує одночасне включення одного символу. Якщо реалізація дозволяє помістити кілька символів у рядку, то функції зчитують прочитають їх у зворотному порядку.

2 Функція `int fflush(FILE *fp)`

Виклик функції `fflush(FILE *fp)` змушує записати в `s` і незбережені дані з буфера у вихідний файл, ідентифіцируємий `fp`. Цей процес називається *очищенням буфера*. Якщо показчик `fp` є нульовим, то очищаються всі буфери.

3 Функція `int setvbuf(FILE *fp, char *buf, int mode, size_t size)`

Функція `int setvbuf(FILE *fp, char *buf, int mode, size_t size)` встановлює альтернативний буфер для використання функціями стандартного введення/висновку. Вона викликається після того, як файл відкритий і перед тим, як над потоком буде зроблена яка-небудь інша операція. Показчик `fp` ідентифікує потік, а перемінна `buf` указує на масив, що буде використаний для збереження. Якщо значенням `buf` не є `NULL`, необхідно створити буфер. Наприклад, можна оголосити масив розміром 1024 символу і передати його адресу. Однак, якщо `buf` має значення `NULL`, те функція зарезервує буфер самостійно. Перемінна `size` повідомляє функції `setvbuf()`, наскільки великим повинний бути масив. (Тип `size_t` є похідним від цілочисельного). Перемінна `mode` може приймати значення: `_IOFBF` — для цілком буферизованого введення/висновку, `_IOLBF` — для лінійно буферизованого і `_IONBF` — для не буферизованого. Функція повертає нуль при успішному виконанні в значення, відмінне від нуля, в інших випадках.

Припустимо, що є програма, що обробляє об'єкти даних, що мають розмір по 3 тис. байтів кожний. Можна використовувати функцію `setvbuf`, щоб створити буфер такого ж розміру.

4 Функції `fread` і `fwrite`

Наступними за списком йдуть функції `fread()` і `fwrite()`, але спочатку потрібно невелика підготовка. Стандартні функції введення/висновку, що використовувалися до цього моменту, орієнтовані на роботу з текстовими файлами, — мають справа із символами і рядками. Що робити у випадку, коли

потрібно зберегти чисельні дані у файлі? Звичайно, можна використовувати функцію `fprintf()` і формат `%f` для збереження значень із крапкою, що плаває, але потім ці значення все рівно зберігаються як рядки, наприклад, послідовність

```
doublenum = 1./3.; fprintf(fp, "%f" , num);
```

збереже `num` як рядок з восьми символів: `0.333333`. При використанні специфікатора `%.2f` зберігаються чотири символи: `0.33`, а при використанні специфікатора `%.12f`— 14 символів: `0.333333333333`. При зміні специфікатора можна залишити більше місця для збереження значення, а також можна зберегти інше значення. Після того як значення `num` збережене як `0.33`, неможливо повернутися до вихідної точності при зчитуванні файлу. Загалом, `fprintf()` конвертує числові значення в строкові, змінюючи їх при цьому.

Найбільш точний і послідовний спосіб запису числа — це зберігати його в такому ж виді, як це робить програма. Отже, значення типу `double` потрібно привласнює перемінної `num` двійкове число `12345`

зберігати в контейнері, що має розмір `double`. Коли дані зберігаються у файлі в такому ж виді, який використовує програма, говорять, що дані зберігаються в двійковій формі. Перетворення числової форми в строкову в цьому випадку не відбувається. У випадку потоку стандартного введення/висновку цю роботу виконують функції `fread`/`fwrite`.

Фактично всі дані зберігаються в двійковій формі. Навіть символи зберігаються в двійковому представленні, що використовує коди символів. Однак, якщо всі дані у файлі інтерпретуються як коди символів, говорять, що файл містить текстові дані. Якщо ж частина даних чи дані в цілому інтерпретуються у виді чисел у двійковій формі, говорять, що файл містить двійкові дані. (Також файли, що являють собою інструкції машинною мовою, є двійковими.)

Необхідно дати деякі пояснення, застосовуючи терміни *двійковий* і *текстовий*. Стандарт ANSI 3 розпізнає два режими відкриття файлів: двійковий і текстовий. Багато операційних систем розрізняють два формати:

двійковий і текстовий. Інформацію можна чи зберігати чи зчитувати у виді тексту двійкових чисел. Усі ці поняття зв'язані, але вони не ідентичні. Текстовий файл можна відкрити в двійковому режимі. Можна зберегти текст у двійковому файлі. Можна використовувати функцію `getc()` для копіювання файлів, що містять двійкові дані. Однак у більшості випадків використовується двійковий режим для збереження двійкових даних у файлі двійкового формату. Подібним чином найбільше часто текстові дані використовуються в текстових файлах, відкритих у текстовому форматі.

5 Функція `size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *fp)`

Функція `size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *fp)` записує двійкові дані у файл. Тип `size_t` визначений за допомогою стандартних типів мови C. Цей тип повертається при виконанні операції `sizeof`. Звичайно мова йде про тип `unsigned`

```
fwrite(&num, sizeof (int), 1, fp);
{
FILE *fa, *fs; // fa для доданого файлу, fs для вихідного файлу
int files = 0; // кількість доданих файлів
char file_app[SLEN]; // ім'я доданого файлу
char file_src[SLEN]; // ім'я вихідного файлу
puts ("Введіть ім'я цільового файлу:");
gets (file_app);
if («fa = fopen(file_app, "a") == NULL)
{
fprintf (stderr, "Неможливо відкрити %s\n", file_app);
exit (2);
}
if (setvbuf(fa, NULL, _IOPBF, BUFSIZ) != 0)
{
fputs ("Неможливо створити вихідний буфер чп", stderr); exit(3);
}
puts ("Введіть ім'я першого вихідного файлу (порожній рядок для виходу) : ");
while (gets(file_src) && file_src[0] != ' \0 ' ) {
if ("strcmp(file_src, file_app) == 0)
fputs ("Неможливо додати файл до самого себе\n", stderr);
elseif ((fs = fopen(file_src, "r") == NULL)
fprintf (stderr, "Неможливо відкрити %s\n", file_src); else {
if (setvbuf(fs, NULL, _IOFBE, BUFSIZ) != 0) {
fputs ("Неможливо створити вихідний буфер хп", stderr);
continue;
}
}
```

```

append (f s , f a) ;
if (terror (fs) != 0)
fprintf (stderr, "Помилка при читанні файлу %s.\n",
file_src) ; if (ferror(fa) != 0)
fprintf (stderr, "Помилка при записі файлу %s.\n",
file_app) ; f close (fs) ; files++;
printf("a^ %s був доданий. \n" , f file_src) ; puts ( "Наступний
файл (порожній рядок для виходу)
}
}
printf ("Готово.  %d файлів додано. \n" , files); f close (fa) ;
return 0 ;
}
voidappend (FILE «source, FILE *dest)
size_tbytes;
externchartemp[]; /* використовуємо зовнішній масив temp */
while ((bytes = f read (temp ,sizeof (char) , BUFSIZE, source) )
> 0)
f write (temp, sizeof (char), bytes, dest) ;
}
Наступний код створює буфер довжиною 1024 байта для кінцевого
файлу: if (setvbuf(fa, NULL, _IOFBF, BOFSIZE) != 0)
{
fputs ("Неможливо створити вихідний буфер\n" , stderr);
exit(3) ;

```

Якщо setvbuf() не може створити буфер, то вона повертає ненульове значення і робота програми припиняється В цьому випадку встановлюється 1024-байтовий буфер для файлу, що копіюється в даний момент. Використання NULL у другому аргументі функції setvbuf() дозволяє функції розподіляти сховище для буфера.

Наступний код запобігає додаванню файлу до самого себе:

```

if (strcmp(file_src, file_app) == 0)
fputs ( "Неможливо додати файл до самого себе\n" , stderr ) ;

```

Аргумент file_app визначає ім'я кінцевого файлу, а file_src — ім'я файлу, оброблюваного в сучасний момент часу.

Функція append() робить копіювання. Замість того, щоб копіювати один байт за один раз, використовуються функції fread і fwrite() для копіювання 1024 байтів за один раз:

```

voidappend (source, dest)
FILE «source, *dest;
{
size_tbytes;

```

```
extern char temp[] ;  
while ((bytes = f read (temp, sizeof (char) , BUFSIZE, source)  
) > 0)  
    f write (temp, sizeof (char) , bytes, dest) ;  
}
```

Оскільки файл, заданий перемінної `dest`, відкритий у режимі додавання, кожен вихідний файл додається в його кінець один за іншим.

У прикладі використовуються текстові файли; при використанні режимів "ab" і "rb" можуть оброблятися також і двійкові файли.

Завдання для самоконтролю знань студентів

- 1 Напишіть програму, що на змінному диску комп'ютера (диск E:) створює файл `numbers.txt` і записує в нього 5 уведених користувачем цілих чисел. Перегляньте за допомогою редактора тексту, наприклад, убудованого в NortonCommander, створений файл. Переконайтеся, що кожне число перебуває в окремому рядку.
- 2 Напишіть програму, що дописує у файл `E:\numbers.txt` п'ять уведених користувачем цілих чисел. Переконайтеся за допомогою редактора тексту, що у файлі перебувають 10 чисел.
- 3 Напишіть програму, що виводить на екран уміст файлу `A:\numbers.txt`.
238. Напишіть програму, що обчислює середнє арифметичне чисел, що перебувають у файлі `A:\numbers.txt`.
- 4 Напишіть програму, що дозволяє переглядати текстові файли (виводить на екран уміст файлу), наприклад, файли вихідних програм C++. Ім'я файлу, що переглядає, повинне передаватися програмі як параметр, у командному рядку під час її запуску.
- 5 Напишіть програму, що дописує в файл `A:` файл `phone.txt` ім'я, прізвище й номер телефону, наприклад, вашого товариша. Якщо файлу на диску ні, то програма повинна створити його. У файлі кожен елемент даних (ім'я, прізвище, телефон) повинен перебувати в окремому рядку.

Критерії оцінювання роботи студента над темою для самостійного вивчення.

В результаті роботи над темою для самостійного вивчення студент повинен надати письмову роботу, яка містить:

- конспект теми;
- відповіді на питання для самоконтролю знань студентів, надані в плані самостійного вивчення теми.

Згідно з розробленою Модульно – рейтинговою системою оцінювання успішності студентів з дисципліни Програмування, максимальна кількість балів, яку може отримати студент за роботу над темою самостійного вивчення, дорівнює 2.

Критерії оцінювання та система нарахування балів за виконання завдань наведено у таблиці № 1:

Таблиця № 1

<i>Види завдань</i>	<i>Кількість балів</i>	<i>Критерії оцінювання виконання завдання</i>	<i>Максимальна кількість балів</i>
Конспект теми	1	Конспект теми повний, логічно-послідовний. Студент зумів виділити та опрацювати основні положення теми, виділені в плані самостійного вивчення теми.	1
	0,75	Конспект повний, але при цьому допущені невеликі пропуски, що не спотворили логічного і інформаційного змісту теми.	
	0,5	Конспект неповно або непослідовно розкриває зміст матеріалу теми, але в конспекті виділені основні положення теми.	
	0,25	Конспект не розкриває основний зміст навчального матеріалу, або студент не зумів виділити та опрацювати основні положення теми.	

Таблиця № 1 (Продовження)

<i>Види завдань</i>	<i>Кількість балів</i>	<i>Критерії оцінювання виконання завдання</i>	<i>Максимальна кількість балів</i>
Відповіді на контрольні питання	1	Студент правильно та повному об'ємі відповідає на всі контрольні питання, надані в плані самостійного вивчення теми.	1
	0,75	Студент правильно та повному об'ємі відповідає на 70% контрольних питань, наданих в плані самостійного вивчення теми.	
	0,5	Студент правильно та повному об'ємі відповідає на 50% контрольних питань, наданих в плані самостійного вивчення теми.	
	0,25	Студент правильно та повному об'ємі відповідає не більше ніж на 30% контрольних питань, наданих в плані самостійного вивчення теми.	
Разом			2

Відповідність кількості набраних студентом балів оцінці за 4 – бальною шкалою оцінювання наведена в таблиці № 2.

Таблиця № 2

Бальна	Національна
1,8-2	відмінно
1,5-1,7	добре
1,2-1,45	задовільно
0-1,2	незадовільно