ЗМІСТ

ВСТУП

РОЗДІЛ 1. ЗНАЙОМСТВО З МОВОЮ С

РОЗДІЛ 2. КЕРУЮЧІ ІНСТРУКЦІЇ

РОЗДІЛ 3. МАСИВИ, ВКАЗІВНИКИ ТА ДИНАМІЧНЕ ВИДІЛЕННЯ ПАМ'ЯТІ

РОЗДІЛ 4. ФУНКЦІЇ

РОЗДІЛ 5. СТРУКТУРИ ТА ОБ'ЄДНАННЯ

РОЗДІЛ 6. ПРИКІНЦЕВІ ЗАУВАЖЕННЯ

Вступ

Мова С та структурне програмування

Усі люди поділяються на дві категорії: тих, хто знає мову С, та тих, хто не вміє програмувати.

Книгу присвячено мові програмування С. Саме С, а не С++. Відмінність у назвах невелика, але яка величезна відмінність у базових підходах! Мається на увазі не лише синтаксис мов С і С++ (хоча тут теж є про що поговорити), але й сама концепція програмування.

Загальноприйнятою є думка, що мова програмування С++ є розширенням мови С для підтримки парадигми об'єктно-орієнтованого програмування (ООП). Цей контекст позиціонує мову С як підмножину мови С++ і багато в чому є справедливим. Але не все так просто. У якийсь момент мова С++ насправді була прямим розширенням мови С, але потім їхні дороги розійшлися. І тенденції такі, що вже навряд чи зійдуться. Як би там не було, уявлення про мову С як про "неповну" або "не до кінця укомплектовану" мову С++ є некоректним. Мова С абсолютно самостійна мова програмування, зі своїми перевагами та недоліками, і зрозуміло, абсолютно самодостатня. Вона проста, лаконічна та ефективна. Мова С однозначно виправдає час та зусилля, витрачені на її вивчення.

Мови та стандарти

Інтелігентна людина має знати кілька іноземних мов. Ну, чи хоча б кілька мов програмування.

В історії розвитку мов С та С++ є певні знакові віхи. Визначаються вони появою нового стандарту мови. Адже що таке, за великим рахунком, мова програмування? Це набір синтаксичних правил, яким повинен відповідати програмний код, плюс компілятор, що дозволяє перетворити програмний код на зрозумілі для комп'ютера інструкції.

До уваги

Загалом у цьому визначенні можна було б обмежитися "набором правил", проте без компілятора весь процес втрачає сенс. Справді, який сенс писати програмний код, якщо комп'ютер його не розуміє і, в принципі, не може зрозуміти? Очевидно, що немає сенсу. Тому за кожною мовою програмування стоїть певна технологія, пов'язана з обробкою та імплементацією програмних кодів. Це такий величезний айсберг, і мова, як набір формальних правил, уособлює лише його вершину.

Що ж до синтаксису мови програмування (С чи С++), то синтаксичні правила визначаються нормативними документами, які зазвичай називаються стандартами мови. Іноді (але не часто) стандарти змінюються (ким і як – зараз не важливо). Зрозуміло, що документ змінити легко. Але цього замало. Потрібно, щоб були компілятори, які підтримують нові стандарти мови.

Деталі

Компілятор – спеціальна програма, призначена для перетворення вихідного коду, написаного мовою на кшталт С або С++, на послідовність інструкцій, які власне і виконуються комп'ютером. Зазвичай після того, як програму написано, її потрібно скомпілювати. Запускається програма на виконання лише після успішної компіляції.

До уваги

Існують мови програмування (наприклад, Python), у яких програми не компілюються, а інтерпретуються. Принципова відмінність інтерпретованих програм від компільованих полягає в тому, що при інтерпретації "перетворення" команд програми у виконавчий код відбувається команда за командою, одночасно з їх виконанням. Що ж до мов С та С++, вони належать до компільованих мов. Зокрема, програма, написана мовою С, компілюється у виконавчий файл, який можна запустити на виконання. Виконується виконавчий файл під керуванням операційної системи.

Стандарт мови єдиний. А от компілятори розробляються різними компаніями. Після зміни стандарту мови розробникам компіляторів необхідно вдосконалити своє програмне забезпечення. Це процес нешвидкий. Тому далеко не всі компілятори повністю підтримують останній (а, отже, і єдино правильний) стандарт мови програмування. З іншого боку, не всі фірми-розробники ставлять перед собою завдання суворо дотримуватись стандарту. Тому можливі ситуації, коли той чи інший компілятор має деякі "особливості".

Переходячи до питань приземлених, тих, що стосуються безпосередньо мови програмування С, зазначимо, що є кілька історично значимих стандартів цієї мови. Важливим став стандарт С89 мови від 1989 року. Потім був стандарт С99 від 1999 року, стандарт С11 від 2011 року, стандарт С17 від 2018 року, та стандарт С23 від 2024 року. У чому особливість стандарту С89? Це своєрідна точка біфуркації, коли мова С, з одного боку, увійшла до складу мови С++, а з іншого, продовжила своє незалежне існування. Те, що сталося з мовою С після стандарту С89, мало пов'язано з розвитком мови С++. Інакше кажучи, зміни до мови С вносилися незалежно від змін до мови С++ (яка теж періодично модифікується).

До уваги

Звичайно, деяка кореляція у процесах удосконалення мов С та С++ існує. Але ті, хто вивчав математичну статистику, знають, що кореляція не обов'язково означає наявність функціонального зв'язку.

Отже, мова С++ є розширенням мови С стандарту С89 з урахуванням тих змін, які вносилися вже безпосередньо в мову С++. Що стосується мови С, то у неї власна доля. І "кооперація" з мовою С++ на ранніх етапах розвитку не скасовує її унікальності та самобутності.

До уваги

На момент написання книги останнім для мови С++ є стандарт С++23 від 2024 року. Він замінив стандарт С++20 від 2020 року. До цього був стандарт С++17 від 2017 року. Попередні стандарти цієї мови С++14, С++11, С++03 та С++98 відповідно з'явилися у 2014, 2011, 2003 та 1998 році.

Отже, останнім (на момент написання книги) є стандарт С23 мови С від 2024 року. Водночас важливо розуміти, що не все описане та задекларованe у стандарті С23 обов'язково буде реалізовано у кожному компіляторі. Тому вибір "правильного" компілятора є завданням важливим і нетривіальним.

Мова С++

Оскільки шляхи С та С++ розійшлися, між ними існують відмінності. Є фундаментальна відмінність, пов'язана з тим, що С++ підтримує парадигму ООП, а мова С є структурною мовою програмування. Але навіть якщо вивести за дужки цей факт, то відмінності спостерігаються вже на базовому рівні. Зрозуміло, тут виявляються наслідки "незалежного" існування мов. У випадках, коли відмінності між С та С++ особливо суттєві, ми використовуватимемо відповідні коментарі у спеціальному блоці. Це важливо не лише задля поінформованості, але ще й через те, що для багатьох програмістів мова С стає трампліном для вивчення мови С++. Тому розумно відразу виділити "тонкі моменти", пов'язані з невідповідністю стандартів цих мов.

Парадигми програмування

Немає настільки шаленої ідеї, яку не можна було б реалізувати у вигляді програмного коду.

Ми вже згадували про "структурність" мови С та "об'єктну орієнтованість" мови С++. Настав час прояснити ситуацію.

Отже, структурне та об'єктно-орієнтоване програмування (скорочено ООП) – дві різні парадигми програмування. Парадигма програмування фактично визначає спосіб організації програмного коду в програмі. Щоб зрозуміти відмінність між структурним програмуванням та ООП, є сенс подивитися на програму як на набір команд або інструкцій, призначених для зберігання, перетворення та обробки даних. За такої інтерпретації програми весь програмний код можна розбити на дві категорії: код, через який реалізуються дані, та код, який призначено для роботи з даними. У межах концепції структурного програмування код для обробки даних реалізується за допомогою процедур та функцій. Програма являє собою набір інструкцій, у яких процедури та функції викликаються для оперування даними. Дані, своєю чергою, зберігаються в змінних, масивах та інших "контейнерах". Дані та код для обробки даних існують незалежно один від одного. Їхня "зустріч" відбувається у програмі. "Місце" та "час" зустрічі визначає програміст, який створює програмний код.

У рамках концепції ООП дані та коди для обробки даних групуються в окремих блоках чи конструкціях, які називаються *об'єктами*. Зазвичай об'єкт

містить і змінні (поля об’єкта), і функції для роботи з ними (методи об’єкта). Програма реалізується як набір взаємодіючих об’єктів.

Якщо порівнювати концепції ООП та структурного програмування, то головну їх відмінність, мабуть, можна було б сформулювати так: якщо у структурному програмуванні зв’язування даних та коду для обробки даних відбувається безпосередньо при виконанні програми, то в ООП таке зв’язування відбувається вже на етапі створення коду для реалізації даних та їхньої обробки.

Концепція структурного програмування з’явилась раніше за концепцію ООП. ООП з’явилося потім. Причини появи ООП прості та прозаїчні. Справа в тому, що зі збільшенням обсягу програмного коду суто технічно стає все складніше поєднувати дані з викликами процедур та функцій. Мовиться про обмежені можливості людини, а не про можливості комп’ютера. Простіше кажучи, що більше за обсягом код, то вища ймовірність для програміста помилитися при його написанні. При використанні концепції ООП за рахунок об’єднання даних та функціонального коду на ранніх етапах проєктування ймовірність помилки зменшується.

До уваги

Написання надійних кодів не означає написання компактних кодів. Використання концепції ООП якщо й спрощує життя програміста, то це зовсім не означає, що програмного коду писати доведеться менше. Радше навпаки.

У структурному програмуванні програма складається з процедур, функцій та змінних (у найширшому сенсі цього поняття). В ООП програма складається з класів та об’єктів.

Деталі

В ООП клас – це певний шаблон, на основі якого створюються об’єкти. Клас містить описання того, що має бути в об’єкті. Якщо провести аналогію і ототожнити об’єкт з будинком, то клас є планом або проєктом такого будинку. Зрозуміло, що за одним проєктом можна звести декілька однотипних будинків. Аналогічно, на основі одного й того самого класу можна створити декілька об’єктів.
Разом з тим, далеко не у всіх мовах, що підтримують парадигму ООП, об’єкти створюються на основі класів. Наприклад, у сценарній мові JavaScript класи як такі відсутні, а об’єкти створюються або "з нуля", або на основі вже існуючих об’єктів. Але це все деталі, які у нашій справі несуттєві.

У мові C++ можна створювати програми як з класами та об’єктами, так і без класів та об’єктів. Простіше кажучи, мова C++ підтримує і парадигму структурного програмування, і парадигму ООП. У мові C підтримується лише парадигма структурного програмування. Тому при створенні програми мовою C ми не зможемо вдатися ні до допомоги класів, ні до допомоги об’єктів. З іншого боку, потреба у такій допомозі виникає набагато рідше, ніж може здатися на перший погляд. Якщо ж мовиться про невеликі програми, то застосування такої "важкої артилерії", як ООП, взагалі видається заняттям малоперспективним. Крім того, в мові C є безліч вбудованих інструментів для написання програм практично будь-якої складності.

До уваги

Як уже зазначалося, мова С++ є перехідною в тому сенсі, що дозволяє вибирати стиль програмування на власний розсуд. Але такий демократизм у світі ООП не є нормою. Радше тут має місце виняток з правил. Скажімо, інші популярні мови програмування Java та C# є повністю об'єктно-орієнтованими. Створення навіть найпростішої програми в Java чи C# вимагає описання щонайменше одного класу.

Що тут важливо? Важливо зрозуміти, що близькість мови С до парадигми структурного програмування насправді не несе ні позитивного, ні негативного навантаження. Просто ось така мова програмування. А те, що це потужна мова програмування, сумніватися не доводиться.

Структура та концепція книги

Найкращий спосіб розібратися в проблемі – написати книгу.

Оскільки книгу присвячено мові програмування С, у книзі описується синтаксис мови і пропонуються приклади програм мовою С. Взагалі, вчитися програмуванню краще та простіше на прикладах. Можна як завгодно довго обговорювати принципи, концепції та підходи щодо створення програм, але якщо все це не підкріплено прикладними навичками, результат буде маловтішним. Матеріал книги підбирався та складався з урахуванням цієї обставини.

Кожен розділ книги присвячено певній темі. Спочатку зазвичай дається теоретична довідка, після якої є невеликі приклади, що стосуються відповідного аспекту чи проблеми. Наприкінці кожного розділу (крім останнього) є приклади виконання завдань, там же є резюме, покликане допомогти читачеві у закріпленні матеріалу розділу. Ще одне призначення резюме – виділити основні, найважливіші моменти, пов'язані з тематикою розділу.

Завершує кожен розділ (крім останнього) список завдань для самостійного вирішення (маються на увазі завдання, які передбачають написання програмного коду). Умови завдань аналогічні до тих програм, що розглядалися у відповідному розділі. Багато в чому завдання для самостійного вирішення є варіацією прикладів, розглянутих раніше. Тому книга може використовуватися для вивчення мови програмування С навіть без викладача, тобто в режимі самонавчання.

Значна частина розглянутих у книзі програм безпосередньо стосуються математики та числових розрахунків. Причин для особливої уваги до математики дві:

- по-перше, завдання, що розглядаються в книзі, самі по собі мають практичний інтерес і часто зустрічаються в рамках різних університетських курсів (насамперед йдеться, звичайно, про числові методи);
- по-друге, алгоритми, використані для вирішення прикладних математичних задач, є показовими щодо застосування можливостей мови програмування С.

До уваги

При розгляді математичних задач, якщо це потрібно, дається коротка довідка щодо суті проблеми. Тож спеціальна математична підготовка читачеві не знадобиться.

Крім математичних завдань, розглядаються й інші. Тому навіть якщо читач не планує надалі присвятити себе вирішенню проблем прикладного математичного характеру, є всі підстави вважати, що книга для нього буде корисною.

Також у книзі увага приділяється порівнянню мов С та С++.

Мова С++

Інформація, важлива для тих, хто планує згодом вивчати мову С++ або вже знайомий з цією мовою, міститься у спеціальних блоках – таких, як цей.

Особлива увага до мови С++ не випадкова. Витоки мови С++, як ми пам’ятаємо, пов’язані саме з мовою С. Але точка зору, згідно з якою те, що має місце в мові С, справедливе і для мови С++, на сьогодні є ілюзією. А ілюзії коштують дорого. Тому ми зробимо все, щоби їх уникнути.

Оскільки в книзі йдеться про програмні коди, а їх потрібно якось тестувати, компілювати та запускати на виконання, зупинимось на особливостях програмного забезпеченням, корисного при написанні програм мовою С.

Компілятор та середовище розробки

Якщо програма компілюється, це вже привід, щоб зробити її комерційною.

На загал, для того, щоб створювати (і запускати на виконання) власні програми, знадобиться як мінімум редактор, в якому набиратиметься програмний код, та компілятор, який перетворить написаний в редакторі код на команди, зрозумілі для комп’ютера. Це мінімальний обов’язковий набір програмних засобів. На практиці зазвичай використовують інтегроване середовище розробки (скорочено IDE від Integrated Development Environment). Це багатофункціональна програма, яка реалізує широкий спектр можливостей в плані розробки, тестування та виконання програмних кодів. На сьогодні існує значна кількість середовищ розробки, які можуть використовуватись в тому числі і при створенні програм мовою С. Огляд усіх цих можливостей не є нашою метою, тому ми зупинимося лише на деяких моментах.

Отже, для програмування мовою С нам щонайменше потрібен компілятор, який "розуміє" мову С. Більшість сучасних компіляторів для мови С дозволяють також компілювати програми мовою С++.

Деталі

Мабуть, правильніше було б сказати, що компілятори для мови С++ дозволяють компілювати програми, написані мовою С. Але це не так вже й важливо. Важливо те, що йдеться про компілятори одразу для двох мов: С та С++.

Існують надійні й функціональні компілятори, які вільно завантажуються і відносно легко встановлюються на комп'ютер. Прикладами є GCC (від GNU Compiler Collection, сторінка підтримки `https://gcc.gnu.org`), MSVC (Microsoft Visual C++, довідка на сайті `https://docs.microsoft.com/cpp`), Mingw-w64 (використовується при роботі з Windows, сторінка підтримки `https://www.mingw-w64.org`) та Clang (використовується при роботі з macOS, Linux та FreeBSD, сторінка підтримки `https://clang.llvm.org`).

Якщо компілятор встановлено, можемо розпочинати програмування. Для цього доведеться набрати програмний код та виконати компілювання. Але якщо ми використовуємо лише компілятор, то компілювати доведеться "вручну", а набирати програмний код у текстовому редакторі. Це цілком допустимо, але дуже незручно. Тому, як уже зазначалося, намагаються використовувати інтегроване середовище розробки. Тут, так само як і з компіляторами, існує низка можливостей (в тому сенсі, що можна використовувати різні середовища розробки). Але мабуть найпопулярнішим (на момент написання книги) є застосунок Visual Studio Code (сторінка підтримки `https://code.visualstudio.com`). Це некомерційне середовище розробки є зручним, ефективним і дозволяє створювати проєкти з використанням різних мов програмування, в тому числі й використовуючи мову С. Воно легко встановлюється і налаштовується для підтримки обраної мови програмування.

Деталі

Що стосується методів роботи з середовищами розробки, то кожне з них має свої особливості. Найкращий спосіб опанувати методи роботи з вибраним середовищем – звернутись до довідки саме з цього середовища. Рекомендації на загал нескладні, але можуть бути й нетривіальні моменти. Наприклад, при роботі з середовищем Visual Studio Code можливо знадобиться запускати його в режимі розробника. Для цього спочатку запускається утиліта Developer Command Prompt, і в терміналі виконується перехід до місця розташування програми: команда **`cd`** (з опцією **`/d`** якщо перехід до іншого диску) та повний шлях до папки з розташуванням програми. Після цього виконується команда **`code .`**, яка запускає середовище Visual Studio Code у потрібному режимі.

Але найпростіший спосіб перевірити коректність роботи власних програмних кодів – скористатись онлайн-компілятором, таким, наприклад, як OnlineGDB (сторінка `www.onlinegdb.com`). Принципи роботи з онлайн-компілятором надзвичайно прості. У відповідному вікні слід вибрати мову програмування (у нашому випадку це мова С) та ввести програмний код. Потім цей код запускається на компіляцію та виконання. Програмні коди з книги перевірялись в основному (але не лише) з використанням онлайн-компілятора OnlineGDB.

Зворотний зв'язок

Листи слід писати. Особливо, якщо їх читають.

Книги пишуться для читачів. Тому важливо знати, які питання читачів турбують насправді. Висловити автору зауваження, побажання та поставити запитання можна електронною поштою `oleksii@vasyliev.kyiv.ua`.

Подяки

Гонорар за це навряд чи заплатять, але на подяку нащадків можна розраховувати.

Автор висловлює щиру подяку своїм студентам та читачам. Їхні відгуки, пропозиції та зауваження сприяють покращенню книг і спонукають рухатись вперед. Інформацію про вже видані книги, а також деякі корисні матеріали, що стосуються цих книг (наприклад, програмні коди прикладів), можна знайти на сайті `www.vasyliev.kyiv.ua` та на YouTube-каналі `www.youtube.com/@oleksii.vasyliev`.

Про автора

Іноді краще вдати, що слова у пісні народні.

Автор книги *Васильєв Олексій Миколайович*, доктор фізико-математичних наук, професор кафедри програмних систем і технологій факультету інформаційних технологій Київського національного університету імені Тараса Шевченка. Автор книг з математичного моделювання та програмування мовами C, C++, C#, Java, JavaScript, Python, PHP. Сфера наукових інтересів: програмування та інформаційні технології, теоретична фізика, біофізика, синергетика, математична економіка, моделювання соціально-політичних процесів, математична лінгвістика.

Розділ 1

Знайомство з мовою С

У наш час не знати мову програмування С просто соромно.

Ми починаємо знайомитись з мовою програмування С. Треба сказати, що мова ця проста, лаконічна, ефективна і в певному сенсі красива. Для початку ми розглянемо кілька простих програм, якими виконуються нескладні дії щодо введення значень з клавіатури та виведення інформації у консольне вікно.

Перша програма

День треба розпочинати з гарної програми.

У будь-якій новій справі важливо правильно ставити запитання. Правильно задане питання іноді важливіше за відповідь на нього. Ми поставимо таке питання: з чого (з яких інструкцій) складається найпростіша програма? Нас цікавить програма, яка робить хоч щось корисне. Невелике, незначне, але таке, що має сенс. А також помітне – тобто таке, що можна помітити, спостерігаючи за виконанням програми. Дуже легко помітити, коли у консольному вікні з'являється повідомлення. Та й операція ця досить проста.

Отже, наша перша програма виводитиме повідомлення. У мові С є вбудована функція `printf()`. Якщо викликати функцію з текстовим аргументом (значення в подвійних лапках), це значення відобразиться у консольному вікні. Тому основна дія, яка виконуватиметься у програмі – це виведення повідомлення (текст у подвійних лапках), вказаного аргументом функції `printf()`. Для більшої конкретики нехай це буде текст `"Hello, World!"`. Інструкція для відображення цього повідомлення має вигляд `printf("Hello, World!")`. Але де слід розмістити відповідну інструкцію? І тут нам знадобляться певні "додаткові" знання.

Існує таке поняття, як *головна функція програми*. Це блок програмного коду з зовнішніми атрибутами функції, який виконується під час виконання програми. Можна сказати й інакше: інструкції, які мають бути виконані під час виконання програми, розміщують у спеціальному блоці, який є головною функцією та називається `main()`. Формальності не такі важливі. Важливо, що у програмі необхідно описати певний стандартний фрагмент чи блок коду.

Шаблон, відповідно до якого описується головна функція програми, має такий вигляд.

```
int main(void){
   // Тут розміщуються команди
   return 0;
}
```

Починається все з ідентифікатора `int`, який у цьому випадку означає буквально наступне: функція `main()` повертає результат, і результат цей є цілим числом. Результат, про який мовиться, повертається інструкцією `return 0` (тобто якщо черга дійде до виконання цієї інструкції, то головна функція завершить роботу та результатом поверне значення `0`).

Деталі

Може здатися дивним, що головна функція програми повертає результат. Проте це не так. Значення, яке повертається головною функцією програми, використовується операційною системою. Якщо програму виконано без помилок, повертається нульове значення. Простіше кажучи, щоб продемонструвати, що програма завершила виконання у "штатному" режимі, функція **`main()`** повинна повернути нульове значення. Для нас це все є формальним моментом. Просто домовимося на майбутнє, що функція **`main()`** завершується інструкцією **`return 0`**. Виклик інструкції **`return`** приводить до завершення виконання функції (у цьому випадку функції **`main()`**). Якщо після інструкції **`return`** вказано значення, воно повертається результатом функції.

Після назви функції у круглих дужках є ключове слово `void`. Ключове слово `void` у круглих дужках після імені функції означає, що при виклику функції аргументи цій функції не передаються.

Мова C++

У мові C++ якщо функція не має аргументів, вона описується з пустими круглими дужками. У мові С пусті круглі дужки не означають відсутність аргументів. Якщо функція не має аргументів, у круглих дужках вказується ключове слово **`void`**.

Команди, що виконуються у програмі, розміщуються у блоці з фігурних дужок. Фактично тут розміщуються саме ті команди, які виконуються під час виконання програми.

До уваги

У програмному коді, крім команд, призначених для виконання, можуть бути ще й коментарі. Коментарі – це текстові пояснення до програмного коду. Коментарі ігноруються під час компіляції програми. Вони призначені виключно для того, хто переглядатиме програмний код. Зазвичай, коментарі містять пояснення щодо тих чи інших команд, блоків команд або містять іншу корисну інформацію.

Існує два способи створення коментарів. Однорядкові коментарі починаються з подвійної нахиленої риски **`//`**. Все, що знаходиться праворуч від подвійної нахиленої риски **`//`**, є коментарем.

Багаторядкові коментарі виділяються символами **`/*`** (початок коментаря) та **`*/`** (закінчення коментаря). Все, що знаходиться між інструкціями **`/*`** та **`*/`**, є коментарем.

У нашому випадку передбачається виконання команди `printf("Hello, World!")`. Ця команда у консольному вікні відображає повідомлення. Усю "роботу" з відображення повідомлення виконує, як зазначалося, функція `printf()`. Але щоб використовувати цю функцію, в програмі необхідно підключити заголовок `<stdio.h>`. Підключення заголовків виконується за допомогою інструкції `#include`, а вся команда виглядає як `#include <stdio.h>` і розміщується першим рядком у програмі перед

описанням функції `main()`. Враховуючи все зазначене, приходимо до висновку, що наша перша програма могла б мати такий код.

```
#include <stdio.h>
int main(void){
   printf("Hello, World!");
   return 0;
}
```

Деталі

Фактично, ми отримали готовий робочий код програми мовою С. Однак є декілька важливих моментів.

По-перше, якщо використовуються кириличні символи, можуть виникнути проблеми з невідповідністю кодування (редактора коду і консольного вікна), внаслідок чого відображатиметься нечитабельний текст.

По-друге, якщо результат програми відображається не у області виведення середовища розробки, а у консольному вікні (це залежить від того, в якому режимі запускається програма на виконання), консольне вікно може закриватись настільки швидко, що відстежити результат виконання програми буде складно.

Ці проблеми стосуються не стільки програмування, скільки особливостей операційної системи та програмного забезпечення. На такому ж рівні їх можна вирішувати. Однак існує прагматичніший підхід. Полягає він у тому, що через програму до консольного вікна передаються спеціальні команди (їх дві):

- одна команда змінює кодування консолі;
- інша команда запобігає згортанню консольного вікна після завершення виконання програми.

Щоб "повідомити" консолі певну команду в програмі використовується функція **`system()`**. Аргументом функції передається текст (вираз у подвійних лапках). Текст містить команду, яка виконується у консольному вікні. Зокрема, щоб запобігти закриванню консольного вікна, виконується команда **`pause`**. Задати для консолі стандартне кодування операційної системи Windows допоможе, наприклад, команда **`chcp 1251`** (інструкція встановити кодову сторінку Windows), **`chcp 866`** (інструкція встановити кодову сторінку DOS) або **`chcp 65001`** (інструкція встановити кодову сторінку UTF-8). Але це ще не все. Виконання команди **`pause`** або команди **`chcp 1251`** (чи інших команд) у консольному вікні призводить до появи повідомлень на кшталт **`Для продовження натисніть будь-яку клавішу...`** чи **`Поточна кодова сторінка: 1251`** відповідно. Простіше кажучи, у консольному вікні з'являються повідомлення, які не пов'язані з виконанням програми. Щоб цього уникнути, при виконанні команд на кшталт **`pause`** чи **`chcp 1251`** додається інструкція, яка "перенаправляє" виведену інформацію в неіснуючий файл, внаслідок чого при виконанні зазначених команд в консольному вікні ніякі додаткові повідомлення не з'являються. Щоб досягти бажаного ефекту наприкінці відповідної команди необхідно через символ **`>`** додати інструкцію **`nul`**: йдеться про виконання команд **`chcp 1251>nul`** та **`pause>nul`**. Саме ці вирази в подвійних лапках передаються аргументами функції **`system()`**, яка, своєю чергою, викликається у програмі. Описання функції **`system()`** міститься в заголовку **`<stdlib.h>`**, який підключаємо до програми. При цьому програмний код міг би бути таким:

```
#include <stdio.h>
#include <stdlib.h>
int main(void){
   // Задаємо кодування для консольного вікна:
   system("chcp 1251>nul");
   // Відображення повідомлення:
   printf("Вивчаємо мову С");
```

```
    // Затримка закривання консольного вікна:
    system("pause>nul");
    return 0;
}
```

У такому випадку у консольному вікні має з'явитись повідомлення **Вивчаємо мову С**.

Наприклад, за потреби (при проблемах з використанням кириличних символів) на початку коду головної функції можна розмістити інструкцію **system("chcp 65001>nul")**. Вона означає, що для відображення символів у області виведення слід використовувати кодування UTF-8. Оскільки функція **system()** належить до бібліотеки **<stdlib.h>**, також слід додати в програму інструкцію підключенні цієї бібліотеки.

Остаточно маємо наступну першу програму, код якої подано в лістингу 1.1.

Лістинг 1.1. Перша програма

```
#include <stdio.h>
int main(void){
   // Відображення повідомлення:
   printf("Hello, World!");
   return 0;
}
```

До уваги

Усі команди у програмі (у головній функції) закінчуються крапкою з комою.

Результат виконання програми такий.

Результат виконання програми (з лістингу 1.1)

```
Hello, World!
```

Отже, найближчим часом наші програми створюватимуться відповідно до наступного шаблону.

```
#include <stdio.h>
int main(void){
   // Тут розміщуються команди
   return 0;
}
```

Цей шаблон має сенс "взяти до уваги". Зрозуміло, що далі ми його ускладнюватимемо, додаючи нові блоки. Тим не менш, вище подано ту "основу", навколо якої "наростає" програма.

До уваги

Декілька слів про описання функції **main()**. У літературі та на довідкових ресурсах можна зустріти різні способи описання цієї функції: наприклад, без ключового слова **void** у круглих дужках або без ідентифікатора типу **int** перед назвою функції. У принципі, програма з таким описанням функції **main()** компілюватиметься. Річ у тім, що хоча головна функція програми повинна повертати результатом ціле число, у більшості компіляторів спосіб описання головної функції програми "сприймається" більш ніж демократично. До цього питання ми ще повернемося, коли обговорюватимемо функції.

Змінні та базові типи

Типи бувають різними. У тому числі й базовими.

У більш-менш серйозній програмі не обійтися без *змінних*. Змінна – це область у пам'яті, до якої можна звертатися на ім'я. Таке ім'я, власне, і ототожнюється зі змінною. Використання змінної в програмі означає звернення до відповідної області пам'яті для зчитування значення, записаного там, або для записування нового значення в цю область пам'яті.

Перед використанням змінну необхідно оголосити. Під час оголошення вказується ім'я змінної та її тип. Тип змінної вказується за допомогою спеціального ідентифікатора типу. Він визначає характер записуваного в змінну значення і, найголовніше, розмір пам'яті, яка виділяється під змінну. У мові С існує всього чотири базові типи даних (якщо не брати до уваги модифікаторів типів), і за великим рахунком всі ці типи числові. Отже, до базових типів мови С належать:

- дійсні числа (ідентифікатор типу `float`);
- дійсні числа подвійної точності (ідентифікатор типу `double`);
- цілі числа (ідентифікатор типу `int`);
- символи (ідентифікатор типу `char`).

До уваги

Крім перелічених типів, до роботи з логічними значеннями, починаючи зі стандарту мови від 1999 року (стандарт С99), додається тип **`_Bool`**. Змінна такого типу насправді є цілим числом. Однак множина її можливих значень обмежена числами **`1`** та **`0`**. Значення **`1`** відповідає істині, а значення **`0`** відповідає хибі.

Взагалі ж як логічні значення в мові С використовуються числа. Будь-яке відмінне від нуля число інтерпретується як істина, а нульове значення інтерпретується як хиба.

Мова C++

У мові С++ для роботи з логічними значеннями є спеціальний тип даних **`bool`**. Змінна типу **`bool`** може приймати два значення: **`true`** (істина) та **`false`** (хиба). У мові С якщо підключити заголовок **`<stdbool.h>`**, можна використовувати псевдонім (альтернативна назва) **`bool`** для типу **`_Bool`**. Також можна буде використовувати ідентифікатори **`true`** та **`false`** як значення для змінних, оголошених з ідентифікатором **`bool`**.

З перерахованих типів три однозначно є числовими. Окремих пояснень заслуговує лише тип `char`. Змінна типу `char` значенням приймає один символ, наприклад, `'A'` або `'Z'` (літерали типу `char` беруться в одинарні лапки). Однак слід мати на увазі, що тип `char` насправді мало чим відрізняється від цілочислового типу (хіба що відрізняється розміром пам'яті). Фактично базові операції з даними цього типу виконуються на рівні кодів символів. Простіше кажучи, ситуація така: кожен символ визначається кодом з кодової таблиці. Тому за фактом те, що ми сприймаємо як символ, насправді є числовим кодом.