## СТРУКТУРНА ОРГАНІЗАЦІЯ ПРОГРАМИ

Сучасні програми містять тисячі рядків коду й використовують різноманітні бібліотеки. Вони настільки великі, що під час написання просто необхідно розбивати їх на окремі частини, які можна було б окремо розробляти, компілювати, тестувати, а потім з них будувати програму. Природно, що ці частини мають бути записані в різних файлах, які компілюються окремо, а потім компонуються в єдину програму.

### 9.1. Програма в кількох файлах

У кожній програмі цієї книги зустрічалася директива препроцесора **#include <iostream>**, яка підключає стандартний бібліотечний файл **iostream** із засобами роботи зі стандартними потоками тощо. Виникає природне питання, чи можна створити власний файл з означеннями певних функцій і так само використовувати його в багатьох різних програмах. Мова програмування C++ має такі засоби!

Спочатку розглянемо, що саме містить заголовний файл **iostream** і як це все працює. За директивою **#include <iostream>** препроцесор додає до тексту програми вміст файлу **iostream** (він зазвичай міститься в підкаталозі **include** каталогу із системою програмування). Вміст файлу **iostream** – це *оголошення* імен змінних, бібліотечних функцій та інших елементів програми, необхідних для використання потоків. Ці оголошення дозволяють компілятору перетворити інструкції програми до машинних кодів, тобто побудувати *об'єктний код*.

Для того, щоб створити *виконуваний код*, до програми необхідно додати, серед іншого, коди бібліотечних функцій, що викликаються в програмі. Ці коди містяться у відповідних бібліотечних файлах, що також входять до складу системи програмування. Створити виконуваний код – задача компонувальника. Отже, як для компіляції програми зазвичай потрібно кілька вхідних файлів, так і для компонування – кілька об'єктних. Цими файлами можуть бути не лише файл із головною функцією та файли зі складу системи програмування, але й інші файли, створені програмістом.

Проілюструємо створення програми, складеної з кількох вхідних файлів, на простому прикладі. У підрозд. 5.3 (див. с. 94) наведено програму з функціями введення й виведення координат точки площини, які викликаються в головній функції. Ці функції роботи з точками можуть знадобитися в інших програмах, але дублювати їх у кожну нову програму зовсім не обов'язково.

Розглянемо спосіб, яким можна уникнути дублювання функцій у програмах. **Перенесемо** функції введення й виведення координат точки площини (а також додамо необхідні для їх успішної компіляції директиви) у новий файл з ім'ям **points.cpp.**

**#include <iostream> using namespace std;**

**void inPoint(double &x, double &y){**

**... }**

**void outPoint(double x, double y){**

**...**

**}**

**points.cpp**

Проте після перенесення імена **inPoint** і **outPoint** у залишку програми не оголошено, тому використовувати їх не можна. Виправимо це. Запишемо прототипи перенесених функцій в окремий заголовний файл (з ім'ям **points.h**).

**void inPoint(double &x, double &y);**

**void outPoint(double x, double y); points.h**

Файл **points.cpp** залишимо без змін. У файлі з головною функцією додатково вкажемо включення файлу **points.h** і збережемо його як **prog022.cpp.**

**#include <iostream> #include "points.h" using namespace std; int main(){**

**… }**

Отже, маємо три файли. Файл **points.h** містить заголовки функцій роботи з координатами точок, файл **points.cpp** – самі функції й необхідні стандартні директиви, а файл **prog022.cpp** – включення **points.h** і головну функцію.

Файл зі складу системи програмування в директиві **include** вказується в кутових дужках як **<iostream>**, а файл, створений програмістом – у лапках як **"points.h".** Кутові дужки вказують, що препроцесор має шукати заголовний файл, починаючи з підкаталогу **include** каталогу із системою програмування та його підкаталогів, лапки – з каталогу з поточним **cpp**-файлом[[1]](#footnote-1). Отже, файли **points**.**h** та **prog022.cpp** краще розмістити в одному й тому самому каталозі.

Зазвичай у заголовних файлах записують оголошення імен функцій, змінних та інших елементів програми.

Для кожного **cpp**-файлу, що містить функції, потрібні в різних частинах програми (або в різних програмах), зазвичай створюють окремий **h-**файл. (У прикладі за файлом **points.cpp** створено **points.h**.) У цей **h-**файл записують оголошення імен функцій і змінних, які мають використовуватися за межами **cpp**файлу. При цьому деякі допоміжні функції або змінні в заголовному файлі можуть бути не оголошені.

Тексти у файлах **points.cpp** та **prog022.cpp** є **одиницями трансляції** (*translation unit*), або **програмними одиницями**. Кожна одиниця містить послідовність функцій, директив препроцесора та інструкцій оголошень імен. Зазвичай

С++-програма складається з кількох одиниць трансляції.

Кожну одиницю трансляції можна скомпілювати окремо. У нашому прикладі результатом будуть об'єктні файли з розширенням **".obj"** (або **".o"**). Далі за допомогою компонувальника з цих файлів можна зібрати виконуваний код програми. Проте зручніше скористатися засобами системи програмування, призначеними саме для створення програм із багатьох вхідних файлів.

У системі програмування Microsoft Visual C++ засобами меню створимо новий **проект**. Типом проекту виберемо **Win32 Console Application** – консольна програма на платформі **Win32** (узагалі, у меню майстра створення проекту можуть указуватися також інші платформи). За допомогою меню додамо до проекту вхідні файли (**Source** **Files**) **points.cpp** та **prog022.cpp**. Далі залишається побудувати виконуваний код (він матиме ім'я проекту з розширенням **".exe"**).

Список файлів проекту ведеться системою програмування в спеціальному файлі (його ім'я та зміст залежать від конкретної системи). Це значно полегшує розробку програми з кількох файлів, особливо якщо їх багато.

#### Вправи

9.1. Модифікувати проект прикладу: додати до файлів **points.cpp** та **points.h** функцію **dist** обчислення відстані між двома точками (див. п. 5.2.1, с. 88) і переробити головну функцію так, щоб програма запитувала в користувача координати двох точок площини й виводила відстань між ними.

9.2. Створити проект і виконувану програму для задачі 6.1 у спосіб, описаний у цьому підрозділі.

### 9.2. Означення й оголошення імен змінних

У п. 5.2.2 на с. 91 розглядалися означення й оголошення функцій. Нагадаємо: **оголошення** повідомляє про наявність імені та його типу, а **означення** ставить у відповідність імені певну ділянку пам'яті (змінна зберігає значення певного типу, функція – опис виконуваних дій). За правилами мови, *кожне оголошене ім'я має бути означеним* (у деякій програмній одиниці проекту). Розглянемо означення й оголошення змінних детальніше.

Інструкція **int x;** не тільки оголошує, але й означає ім'я **x**, адже за цією інструкцією утворюється змінна **x**, тобто для імені **x** призначається ділянка пам'яті. Наявність чи відсутність ініціалізації змінної тут ні на що не впливає: усе рівно це означення.

Коли програма складається з кількох файлів, може виникнути необхідність у глобальних змінних, що використовуються в різних одиницях трансляції. Проте змінна не може мати більше одного означення! Тому її необхідно *означати* *тільки в одній одиниці трансляції, а в усіх інших – лише оголошувати*.

Оголошення глобальних змінних у мові C++ має вигляд **extern ім'я-типу ім'я-змінної;**

Означенням **int a;** та **int b=3;** відповідають оголошення **extern int a;** та **extern int b=3;**. За правилами мови C++, якщо глобальну змінну означено зі специфікатором **const**, наприклад **const int OK=0;**, то відповідне оголошення теж повинно мати цей специфікатор і те саме ініціалізуюче значення:

**extern const int OK=0;**.

Оголошення глобальних змінних зазвичай записують у заголовному файлі.

За оголошенням змінної, яке не є означенням, окрема ділянка пам'яті під змінну не виділяється.

**Вправа 9.3**. Модифікувати програму **prog021.cpp** (с. 144), виділивши функцію **get** в окрему одиницю трансляції, і створити для неї проект і виконувану програму в спосіб, описаний у цьому підрозділі.

### 9.3. Поняття простору імен

У цьому підрозділі викладено лише початкові відомості про простори імен у мові C++ (детальніше див. бібліографічний список).

Почнемо з прикладу. Відомо, що в метричній системі префікси "кіло" й "мега" задають множення на 103 та 106, але у випадку вимірювання інформації програмісти часто використовують ці префікси для множення на 210 та 220. Нехай є **cpp-**файли (і відповідні **h-**файли), кожен з яких містить функції **kilo** та **mega** для перетворення одиниць вимірювання "кіло" й "мега" на прості одиниці. Відмінність між цими двома файлами лише в тому, що в першому функції виконують множення на відповідний степінь числа 10, а в другому – на степінь 2. Припустимо, що в програмі необхідні функції з обох файлів. Спроба включити обидва файли в проект буде неуспішною, оскільки функція **kilo** має в проекті два означення (у таких ситуаціях кажуть, що виник *конфлікт імен*). Уникнути цих проблем можна, якщо скористатися *простором імен* (*namespace*).

**Простір імен** – це іменована частина програми, що містить оголошення, означення та інші елементи мови.

Простір імен має такий синтаксис:

**namespace *ім'я-простору* { *вміст простору імен, тобто оголошення, означення тощо***

**}**

Ім'я, оголошене всередині простору імен, отримує додатковий ідентифікатор – ім'я простору. Наприклад, нехай записано простір імен **mySpace**. **namespace mySpace { int name; }**

Ім'я **name**, оголошене всередині простору, для компілятора задається *глобальним* ідентифікатором **mySpace::name**; його перша частина вказує *контекст* (тут це простір імен), друга – *локальне* ім'я в межах контексту (простору імен). Оператор :: на-

зивають **оператором розв'язання контексту** (*scope resolution operator*). Неформально вираз **mySpace::name** можна розуміти як "ім'я **name**, оголошене в межах **mySpace**". Результатом застосування оператора розв'язання контексту є **кваліфіковане ім'я** (*qualified*, з англ. – кваліфікований, специфікований,уточнений). У межах простору **mySpace** діють як локальне ім'я **name**, так і кваліфіковане **mySpace::name**, але за межами простору – лише кваліфіковане **mySpace::name**. Іншими словами, областю дії оголошення імені **name** є простір **mySpace**, тому за його межами оголошення імені **name** діяти й не повинно.

Завдяки тому, що область дії оголошення імені обмежено простором імен, однакові імена, оголошені в різних просторах, стають для компілятора різними. Саме це й дозволяє уникати конфліктів імен.

Повернемося до задачі про одиниці вимірювання. Уведемо два простори імен функцій: **metric** – для обчислень за метричною системою, **binary** – за двійковою системою. Функції простору **binary** оголосимо у файлі **prefix2.h**. **namespace binary { double kilo (double x);//\*1024 double mega (double x);//\*(1024\*1024) }**

У файлі **prefix2.cpp** запишемо означення функцій (також у просторі).

**namespace binary { double kilo (double x)**

**{ return x\*1024;} double mega (double x) { return x\*1048576;}**

**}**

Так само у файлі **prefix10.h** оголосимо функції простору **metric**.

**namespace metric{ double kilo (double x);//\*1000 double mega (double x);//\*(1000\*1000)**

**}**

У файлі **prefix10.cpp** запишемо їх означення.

**namespace metric{ double kilo (double x)**

**{ return x\*1000;} double mega (double x) { return x\*1000000;}**

**}**

Проілюструємо використання розроблених **cpp**-файлів простою програмою.

**#include <iostream>**

**#include "prefix2.h"**

**#include "prefix10.h" using namespace std; int main(){**

**cout<<metric::kilo(1)<<endl; // 1000 cout<<binary::kilo(1)<<endl; // 1024**

**system("pause"); return 0;**

**}**

Практично всі наведені вище програми містять директиву **using**. **using namespace std;**

Вона задає використання стандартного простору імен **std**. Насправді бібліотечні засоби мови C++ для роботи з потоками, наприклад ім'я **cout**, оголошено в просторі імен **std**. За межами цього простору стандартний потік виведення для компілятора ідентифікується не ім'ям **cout**, а кваліфікованим ім'ям **std::cout**. Отже, інструкція виведення повинна була б мати вигляд **std::cout<<"some** **text";**. Проте директива **using namespace std;** дозволяє далі позначати стандартний потік виведення просто локальним ім'ям: **cout<<"some text";**.

Директиви вигляду **using namespace** ***ім'я-простору*;** забезпечують ще один спосіб уникати конфліктів імен. Областю дії директиви **using namespace**, записаної поза блоками, є текст програми до кінця файлу, в якому її записано. Якщо ж її записано в блоці, то вона діє в межах цього блоку. Звідси в *різних, не вкладених* один в одного, блоках можна вказати використання різних просторів імен, і далі в кожному з блоків записувати імена з відповідного простору, не кваліфікуючи їх. Наприклад, наведену вище головну функцію можна написати так:

**int main(){**

**{ using namespace metric; cout<<kilo(1)<<endl; // 1000**

**}**

**{ using namespace binary; cout<<kilo(1)<<endl; // 1024**

**}**

**system("pause"); return 0;**

**}**

Кожна директива використання простору у своїй області дії, по суті, додає локальні імена цього простору. Якщо між іменами різних просторів, потрібних у програмі, немає конфліктів, то директиви використання всіх цих просторів можна записати на початку програми й далі працювати лише з локальними іменами. Наприклад, якщо в програмі потрібні імена лише з просторів **std** та **binary**, то десь на її початку варто записати обидві директиви.

**using namespace std; using namespace binary;**

Завдяки ним далі в програмі можна користуватися некваліфікованими іменами **cin**, **cout** тощо, а також **kilo** та **mega**.

Досить часто в програмах потрібні лише окремі локальні імена, оголошені в деякому просторі імен. Замість використання всього простору можна оголосити лише потрібні імена, кожне у вигляді **using *ім'я-простору* :: *локальне-ім'я*;**

Наприклад, у більшості програм цієї книжки замість директиви **using namespace std;** можна було б написати два зовнішніх оголошення **using std::cin; using** **std::cout;** або, якщо з наведених вище функцій перетворення в одному блоці потрібні **binary::kilo** та **metric::mega** – то оголосити їх у цьому блоці. **using binary::kilo; using metric::mega;**

Це дозволить звертатися до функцій, не вказуючи просторів імен, яким вони належать.

Усі імена, оголошені поза межами просторів імен, незалежно від їх розташування по одиницях трансляції належать єдиному глобальному простору імен.

Створюючи бібліотеку функцій, призначених для багаторазового використання в різних програмах, варто означити весь код у певному просторі імен. Це може значно полегшити його використання.

**Вправа 9.4**. З використанням розроблених у цьому підрозділі бібліотечних файлів написати програму, що запитує в користувача одиницю виміру (кілобайт або кілограм) і кількість та перетворює отриману кількість на байти чи грами, відповідно. Створити для неї проект і виконувану програму.

#### Контрольні запитання

9.1. Чи може програма мовою С++ складатися з кількох файлів?

9.2. З файлів якого різновиду будує програму препроцесор, а якого – компонувальник?

9.3. Для чого в мові C++ використовуються заголовні файли?

Що варто в них записувати?

9.4. Що таке простір імен?

9.5. Що таке кваліфіковане ім'я?

9.6. У чому полягає суть оператора розв'язання контексту?

#### Задачі

9.1. Підготувати реферат на тему: "Робота з просторами імен у мові C++".

9.2. Підготувати реферат на тему: "Директива й оголошення **using** у мові C++".

### ДОДАТОК А Системи числення

**Поняття системи числення**

**Система числення** – це система правил запису чисел.

Людина звикла до десяткової системи числення, що з'явилася в стародавніх Вавилоні та Індії, потім стала відома арабам і завдяки їм потрапила до Європи. У цій системі є *десять знаків* – цифр, якими позначають числа від **0** до **9**. Більші числа позначають *послідовностями* цифр. Знаки в послідовності займають різні **позиції** (**розряди**): цифра праворуч позначає кількість одиниць, наступна – кількість десятків тощо. Цифра залежно від позиції задає різні значення (у записі **32** цифра **2** – дві одиниці, а в **23** – два десятки). Цю систему запису чисел називають **позиційною**.

Кількість знаків у системі числення називається її **основою**.

Люди користуються записом чисел з основою не тільки 10, але й, наприклад, 12, 60 або 5. У програмуванні можна зустріти запис чисел з основою 8 та 16.

У десятковій системі цифра праворуч у записі числа називається **молодшою**, ліворуч – **старшою**. Розряд одиниць називається нульовим, десятків – першим, сотень – другим тощо. Розряд указує, на який *степінь числа 10 треба помножити цифру* в цьому розряді (одиниці – на 100, десятки – на 101, сотні – на 102 тощо). Отже, число записується як сума добутків цифр числа на відповідні степені десятки, наприклад, 5834 5103+8102+3101+4100.

Якщо запис числа має дробову частину, то додаються *цифри, ділені на 10 у відповідних степенях*, наприклад,

0,234 210-1+310-2+410-3.

Так само числа записуються з іншими основами більше 1. Питання лише в тому, які знаки є цифрами. Якщо основа не більше 10, то беруть звичні десяткові цифри (стільки знаків, скільки треба). Проте за основи більше 10 потрібні додаткові знаки, щоб позначати десяткові числа 10, 11, …. У XX ст. для цього стали використовувати послідовні великі літери латинського алфавіту – *A*, *B*, *C* тощо.

#### Приклади

1. У двійковій системі числення лише дві цифри – **0** та **1**. Двійкові записи **10**, **11**, **100**, **101** позначають десяткові 2, 3, 4, 5. Запис **1101** позначає число 123+122+021+120  13, запис

**0,101** – число 12-1+02-2+12-3  0,625.

1. У системі числення з основою 8 для запису чисел є цифри **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, "вага" кожного розряду числа є відповідним степенем вісімки. Отже, перші числа записуються як **0**, **1**, … **6**,

**7**, а далі йдуть записи **10**, **11**, …, **17**, **20**, **21**, …, **77**, **100**, …. Вісімкове **10** – це звичне десяткове 8, **11** – звичне 9, **20** – десяткове 16, тобто двічі по 8. А вісімкові **100**, **200** та **400** – це десяткові 64, 128 та 256, тобто один, два й чотири рази по 8 у квадраті. Так само 3428  382+481+280, 0,3428  38-1+48-2+28-3, 34,28  381+480+28-1 (про вісімковий запис свідчить маленька цифра 8 унизу).

1. У шістнадцятковій системі числа від 10 до 15 позначають так: 10 – **A**, 11 – **B**, 12 – **С**, 13 – **D**, 14 – **E**, 15 – **F**. Тоді **2A**16  2161+10160  42, **FF**16  15161+15160  255,

-1

**0,C**16  1216  ¾ 0,75. 

#### Перетворення запису числа з недесяткової системи на десяткову

У комп'ютері числа зображуються у двійковій системі числення. Проте людині зручніше вводити числа в комп'ютер та отримувати їх від нього як десяткові. Виникає **задача** **перетворення запису числа** з однієї системи на іншу (зокрема, з двійкової на десяткову й навпаки).

Запис (*xn*…*x*1*x*0,*x*-1…*x*-*k*)*P* з *P*-ковими цифрами *xi* задає число *xn**Pn*+…+*x*1*P*1+*x*0*P*0+*x*-1*P*-1+…+*x*-*k**P*-*k*. Цифри *xi* у цій сумі позначають числа від 0 до *P –* 1. Звідси, щоб знайти десяткове зображення числа, потрібно:

**записати в десятковій системі число *P* та *P*-кові цифри числа;**

**обчислити суму добутків значень цифр і відповідних степенів числа *P*.**

##### Приклади

100112  124+023+022+121+120  19;

10,0112  121+020+02-1+12-2+12-3  2,375;

**1BC**16  1162+11161+12160  444;

**1B,C**16  1161+11160+1216-1  27,75;

12,23  131+230+23-1  5,666…, тобто число записується нескінченним періодичним десятковим дробом. 

##### Вправи

А.1. Перетворити шістнадцяткові записи чисел на десяткові:

а) F1; б) FF; в) 4AB; г) FFFE.

А.2. Перетворити вісімкові записи чисел на десяткові:

а) 377; б) 1777; в) 1232; г) 400.

А.3. За основою *P* та *P*-ковими записами дробів *Q* навести їх десяткове зображення:

а) *P* 2; *Q* 0,0001; 0,1111;

б) *P* 3; *Q* 0,22; 0,(11);

в) *P* 16; *Q* 0,1; 0,F; 0,8; 0,(7);

г) *P* 2; *Q* 0,001; 0,1101; 0,000001;

д) *P* 3; *Q* 0,(20); 0,(02); 0,11;

е) *P* 16; *Q* 0,1F; 0,FE; 0,81; 0,3(F).

#### Утворення недесяткового запису цілих чисел

Розглянемо, як за натуральним числом *N* у десятковому записі отримати цифри *P*-кового зображення. Невідомі ані самі цифри, ані їх кількість. Проте відомо, що *P*-ковий запис задає число як суму добутків значень цифр на відповідні степені числа *P*, тобто за деякого невідомого *n* і невідомих цифр *xi* справджується рівність

*N* (*xn*…*x*1*x*0)*P*  *xn**Pn*+…+*x*1*P*1+*x*0*P*0.

Зауважимо: усі доданки, окрім останнього, мають множник *P*. Тоді значенням молодшої цифри *x*0 є остача від ділення *N* на основу *P*, а сума *T* *xn**Pn*-1+…+*x*1*P*0  (*xn*…*x*1)*P* дорівнює цілій частці від ділення *N* на *P*. Поділивши цю суму на *P* з остачею, знайдемо остачу *x*1 і наступну частку і так далі, поки на якомусь кроці частка від ділення не стане рівною 0. Остання остача й буде старшою цифрою *xn*.

**Приклади**. Утворимо вісімковий запис числа 202.

**202 : 8 = 25** (остача **2** – молодша цифра),  **25 : 8 = 3** (остача **1** – наступна цифра).

**3 : 8 = 0** (остача **3** – остання, старша цифра).

Отже, отримано цифри **2**, **1**, **3** (від молодшої до старшої), тобто вісімковий запис **312**.

Запишемо 202 в шістнадцятковій системі.

**202:16=12** (остача **10** – значення молодшої цифри **A**),  **12:16= 0** (остача **12** – значення старшої цифри **С**).

Отже, десяткове 202 має шістнадцятковий запис **CA**. 

Позначимо остачу від ділення *T* на *P* націло як *T* % *P*, частку – як *T* / *P*. Опишемо утворення *P*-кового запису числа *N* за допомогою алгоритму.

**1. Спочатку частка T дорівнює N, а запис порожній. 2. Поки T > 0,**

**обчислити остачу R як T % P і нову частку T як T / P (шляхом ділення в стовпчик), зобразити R у P-ковій системі числення як цифру C, відповідну до R, і дописати цифру C до запису ліворуч.**

**Вправа А.4**. Перетворити десятковий запис на двійковий, вісімковий і шістнадцятковий:

а) 94; б) 768; в) 16382; г) 65537.

#### Утворення недесяткового запису дробових чисел

Розглянемо, як за додатним дійсним числом *V* < 1 отримати цифри його *P*-кового запису (принаймні кілька перших, оскільки запис може бути нескінченним). За способом запису

*V* (0**,***x*-1…*x*-*k*…)*P* *x*-1*P*-1+…+*x*-*k**P*-*k*+… з невідомими значеннями цифр. Помножимо обидві частини рівності на *P* й отримаємо рівність

*V**P* *x*-1+*x*-2*P*-1+…+*x*-*k**P*-*k*+1 +… (*x*-1**,***x*-2…*x*-*k*…)*P*  *x*-1+*P*-1  (0, *x*-2…*x*-*k*…)*P* .

Звідси *x*-1  [*V**P*], а *P*-1(0,*x*-2…*x*-*k*…)*P* {*V**P*}, де [*V**P*] та {*V**P*} позначають цілу й дробову частини *V**P*. Далі так само помножимо {*V**P*} на *P*, знову отримаємо цілу й дробову частини (ціла буде значенням *x*-2) і т. д.

**Приклад**. Утворимо двійковий запис десяткового дробу 0,75.

0,752 1,5, [1,5] **1** (перша цифра), {1,5} 0,5; 0,52 1, [1] **1** (наступна цифра), {1} 0.

Усі подальші цифри будуть **0**, тому **0,11** є скінченним двійковим зображенням для десяткового 0,75.

Утворимо двійковий запис десяткового дробу 0,1.

0,12 0,2, [0,2] **0** (перша цифра), {0,2} 0,2;

0,22 0,4, [0,4] **0** (наступна цифра), {0,4} 0,4;

0,42 0,8, [0,8] **0** (наступна цифра), {0,8} 0,8;

0,82 1,6, [1,6] **1** (наступна цифра), {1,6} 0,6; 0,62 1,2, [1,2] **1** (наступна цифра), {1,2} 0,2.

Далі цифри **0**, **0**, **1**, **1** будуть повторюватися до нескінченності, тобто точний двійковий запис десяткового 0,1 є періодичним **0,0(0011).** Будь-який початок цього запису позначає наближення до десяткового 0,1, наприклад, 0,000112  1/16+1/32 0,09375 (помилка – приблизно шість тисячних).

Утворимо шістнадцятковий запис десяткового дробу 0,8.

0,816 12,8, [12,8] 12 (перша цифра **C**), {12,8} 0,8. Далі цифра **C** буде нескінченно повторюватися, тому **0,(C)** є точним шістнадцятковим записом десяткового 0,8. 

Якщо *V* є скінченним десятковим дробом і основа *Р* має прості множники 2 та 5, то число *V* можна зобразити скінченним *P*-ковим записом. Інакше *P*-ковий запис може бути нескінченним; тоді скінченна кількість кроків дає *наближене зображення* числа *V*.

Отже, за дійсним числом *V*, *V* < 1, можна одержати *R* перших цифр його *P*-кового зображення, виконуючи такий алгоритм:

1. **Спочатку зображенням є "0.".**
2. **Поки одержано менше ніж R дробових цифр, обчислити V****P, обчислити d як [V****P] (ціле число від 0 до P-1) та V як {V****P}. Записати значення d як P-кову цифру й дописати її до зображення праворуч.**

Записи цілих і дробових чисел перетворюються з однієї системи числення на іншу *за* *різними алгоритмами*. Тому, перекладаючи запис числа, що має цілу та дробову частини, треба спочатку виділити ці частини й потім перетворити їх *окремо*.

Узагалі, щоб перекласти запис числа з довільної системи числення на іншу, необхідно виконати багато арифметичних дій у незвичній недесятковій системі. Інколи простіше скористатися десятковою системою як проміжною.

1. Або в додаткових каталогах, які зазначає програміст (висвітлення цих можливостей системи програмування виходить за межі книги). [↑](#footnote-ref-1)