2. Програми з простим повторенням

У цьому параграфі йтиме мова про задачі, у яких необхідно виконати послідовний перебір членів деякої числової послідовності. На прикладах програм попереднього параграфа ми уже демонстрували читачеві, що для побудови алгоритму важливе значення має вибір структури даних. Чи завжди для опрацювання заданої послідовності значень a_1, a_2, ..., a_n у програмі використовують масив? Підсвідомо хочеться відповісти «так», адже в умові задачі записано змінні з індексами. Проте, поспішати з висновками не будемо, бо у постановці задач здебільшого використовують систему математичних позначень, яка не збігається з «системою позначень» програми. Математик пише $y(x)$, щоб вказати функціональну залежність *математичних* величин y та x, а програміст використовує *прості змінні: x* – для зберігання заданої величини та *y* – для запису результату обчислень. А дужки після імені змінної у мовах програмування мають спеціальне синтаксичне значення. Так і масиви у програмах використовують далеко не в кожному випадку, коли в умові задачі є змінні з індексами. Наприклад, якщо кількість членів заданої послідовності наперед невідома, то оголосити масив просто не вдасться. Для перебору значень звичайно застосовують відповідні цикли. Якщо кожне значення враховується тільки на одній ітерації циклу, то для його зберігання доцільно використати просту змінну, а не елемент масиву.

Усі приклади алгоритмів оформимо як функції мовою C++. Головна програма для задач цього параграфу матиме схожий вигляд, як у попередньому: досить лише замінити назви пунктів меню і функцій на відповідні.

2.1. Покрокове введення даних

Часто умову задачі формулюють так, що для отримання результату необхідно певним чином опрацювати кожен член заданої послідовності. Наприклад:

Задача 6. *Задано натуральне число n і дійсні* a_1, a_2, ..., a_n. *Обчислити* $\sqrt{\frac{a_1^2+a_2^2+\ldots+a_n^2}{n}}$.

Щоб отримати розв'язок задачі, необхідно знайти відповіді на кілька запитань. Що є шуканим значенням? Корінь з суми квадратів членів послідовності, розділеної на їхню кількість. Для її накопичення використаємо змінну *sum*. Що необхідно зробити з кожним значенням послідовності $\{a_i\}$? Піднести до квадрату і додати до загальної суми. Ці дії можна виконати протягом однієї ітерації, тому для тимчасового зберігання a_i використаємо просту змінну, наприклад, *a*. Кількість членів послідовності задано значенням *n*. Тому для їх перебору зручніше застосувати цикл з параметром – змінною *i*, оскільки кількість повторень буде задано значенням *n*. Які дії необхідно виконати перед циклом? Тільки ініціалізувати змінну *sum*. Ділення на *n* та обчислення кореня виконаємо після закінчення циклу з додаваннями.

Тепер можемо записати програму:

```
void SquareNorm()
{
	cout << "\n *Середнє квадратичне заданої кількості дійсних чисел*\n"
		<< "\nВведіть натуральне число: ";
	unsigned n; cin >> n;
	double sum = 0.0;
	for (unsigned i = 1; i <= n; ++i)
	{
		cout << "Введіть " << i << "-е число: ";
		double a; cin >> a;
		sum += a * a;
```

```
        }
        sum = std::sqrt(sum / n);
        cout << "Середнє квадратичне = " << sum << '\n';
        return;
}
```

Іноді в задачах вказують не кількість членів послідовності, а умову її закінчення: послідовність чисел закінчується нулем, послідовність літер закінчується крапкою тощо. У таких випадках перебір виконують за допомогою циклу з перед- або постумовою.

Задача 7. *Задано послідовність чисел, яка містить хоча б одне нульове значення, і перший член якої відмінний від нуля. Обчислити середнє арифметичне членів послідовності, що передують першому нулеві.*

Тут умовою закінчення обчислень є рівність нулю чергового члена послідовності. Як і в попередній задачі для зберігання значень послідовності використаємо просту змінну *a*. Для обчислення суми значень і їхньої кількості використаємо змінні *sum* і *quantity*. Ці змінні необхідно ініціалізувати перед початком циклу.

```
void Average()
{
        cout << "\n *Середнє арифметичне послідовності дійсних чисел "
             << "з термінальним елементом*\n\n";
        cout << "Введіть послідовність чисел, що закінчується нулем:\n";
        unsigned quantity = 0;
        double sum = 0.0;
        double a; cin >> a;  // перший член послідовності
        while (a != 0.0)     // чи він не термінальний?
        {
              sum += a;      // накопичення суми та кількості
              ++quantity;
              cin >> a;      // наступний член послідовності
        }
        if (counter == 0) cout << "Послідовність починається нулем\n";
        else cout << "Середнє арифметичне = " << sum / quantity << '\n';
        return;
}
```

За умовою задачі послідовність містить ненульові значення, тому внаслідок виконання програми отримаємо *quantity* > 0 і оператор *sum* / *quantity* виконається без помилок. Для більшої надійності програми ми виконали додаткову перевірку значення *quantity*:

```
        if (quantity == 0) cout << "Послідовність починається нулем\n";
        else cout << "Середнє арифметичне = " << sum / quantity << '\n';
```

Зверніть увагу на спосіб введення даних у функції *Average*. Перший член зчитується ще перед початком циклу (щоб було що порівнювати з нулем в умові циклу). Наступні значення зчитуються в циклі вже після обробки попереднього значення. Отже, буде враховано кожен член послідовності, крім останнього – нуля, як і потрібно за умовою задачі.

2.2. Покрокове виведення даних

До окремої категорії задач належать ті, в яких йде мова не про обробку конкретної послідовності, а про обчислення і виведення на друк певним чином заданих послідовностей. Розглянемо кілька прикладів.

Задача 8. *Протабулювати функції* sin(*x*) *та* cos(*x*) *на проміжку від 0 до* π *з кроком* π/12.
(Це означає отримати таблицю значень функцій для $x = 0, \pi/12, \pi/6, ..., \pi$*).*

У цій задачі необхідно обчислити три послідовності: $x_i = i\,\pi/12$, $s_i = \sin x_i$, $c_i = \cos x_i$ для $i = 0, 1, ..., 12$. Для тимчасового зберігання членів цих послідовностей використаємо прості змінні *x*, *sin_x* та *cos_x* відповідно. Межі зміни індексу послідовностей відомі, тому застосуємо цикл з параметром.

Число π у програмах мовою C++ позначають константою *M_PI*. Щоб скористатись нею, стандартними функціями sin(*x*) та cos(*x*), а також засобами форматування потоку виведення, приєднаємо до тексту програми відповідні заголовкові файли. У коментарях до тексту програми наведено відповідні пояснення.

```
#define _USE_MATH_DEFINES   // приєднуємо визначення математичних констант,
#include <math.h>           // оголошення стандартних математичних функцій
#include <iomanip>          // та маніпуляторів форматами виведення

using std::setw;
using std::setprecision;

void Table()
{
      cout <<
         "\n *Таблиця значень тригонометричних функцій на проміжку [0;Pi]*\n\n";
      const int n = 12;
      // друкуємо шапку таблиці
      cout << "  x     sin x    cos x\n------------------------\n";
      double h = M_PI / n; // крок зміни аргументу досить порахувати один раз
      for ( int i = 0; i <= n; ++i) // обчислюємо і друкуємо рядки таблиці
      {
            double x = h * i;
            double sin_x = sin(x);
            double cos_x = cos(x);
            cout << std::showpoint << std::fixed
                  << setw(5) << setprecision(2) << x
                  << setw(9) << setprecision(5) << sin_x
                  << setw(9) << setprecision(5) << cos_x << '\n';
      }
      return;
}
```

Чергові значення x_i, s_i, c_i нема сенсу зберігати після їхнього надрукування, тому в програмі можна обійтися без масивів, використовуючи лише прості змінні.

Форматом виведення керують такі маніпулятори: *showpoint* наказує друкувати десяткову крапку для всіх дійсних чисел, *fixed* – застосовувати формат з фіксованою крапкою (не з плаваючою), *setw* задає ширину поля виведення, *setprecision* – кількість знаків після крапки.

Задача 9. *Нехай послідовність {a_i} задано співвідношеннями:* $a_0 = 1$; $a_k = \frac{a_{k-1}}{k} + 0{,}2k - 1$, *k = 1, 2, … . Задано m (m>1). Надрукувати a_0, …, a_{n-1}, де n – номер першого члена послідовності, для якого виконується умова $a_n > m$.*

Число *n* наперед невідомо, тому масив оголосити і використати не вдасться, проте він і не потрібен: для обчислення членів послідовності достатньо однієї простої змінної *a*. Обчислення рекурентної формули з умови задачі можна задати такою інструкцією:

```
a = a / k + 0.2 * k + 1;
```

Змінна *a* в правій частині оператора містить значення a_{k-1}. У результаті виконання обчислень отримаємо нове значення, яке є значенням a_k. На наступному кроці воно замінить значення a_{k-1} у змінній *a*:

```
void Succession()
{
	cout << "\n *Побудова зростаючої числової послідовності*\n\n";
	double m;
	cout << "Введіть граничне значення: "; cin >> m;
	int k = 1;
	double a = 1.0; // задали перший член послідовності
	cout << "a(" << k << ")=" << a << '\n';
	while (a <= m)
	{
		++k;
		a = a / k + 0.2 * k + 0.1; // обчислили наступний
		cout << "a(" << k << ")=" << a << '\n';
	}
	cout << "Знайдений номер = " << k << '\n';
	return;
}
```

2.3. Обчислення за рекурентними формулами

Попередній приклад уже містив такі обчислення. Розглянемо складніші випадки.

Задача 10. *Числа Фібоначчі задано рекурентними співвідношеннями:* $f_0 = f_1 = 1$; $f_n = f_{n-1} + f_{n-2}$, *n = 2, 3, … . Обчислити k-е число Фібоначчі (k>0).*

На відміну від попередньої задачі, тут для обчислення чергового члена послідовності потрібні значення не одного, а двох попередніх. Щоб побудувати алгоритм, використаємо такі змінні: *n* – для номера чергового числа Фібоначчі; *f_n_2* – для зберігання значення f_{n-2}; *f_n_1* – для f_{n-1} і *f_n* – для f_n. Очевидно, що для *n* = 2 необхідно виконати

```
f_n_2 = f_n_1 = 1; f_n = f_n_1 + f_n_2;
```

Тут *f_n_1* і *f_n_2* містять «старі» значення, а *f_n* – «нове». Після збільшення *n* значення *f_n* стане «старим», а значення, що міститься в *f_n_2*, – непотрібним для наступних обчислень. Щоб відобразити в алгоритмі «старіння» значень, виконують переприсвоєння:

```
f_n_2 = f_n_1; f_n_1 = f_n;
```

Тепер запишемо цілу програму:

```
void Fibonacci()
{
	cout << "\n *Побудова послідовності чисел Фібоначчі*\n\n";
	unsigned k;
	cout << "Введіть номер (>=2) останнього числа: "; cin >> k;
	k = (k > 2) ? k : 2;	// "підстрахуємо" правильність вхідних даних
	long long f_n;
	long long f_n_1 = 1;	// задали початок послідовності
	long long f_n_2 = 1;
	cout << "f(0)=" << f_n_2 << "\nf(1)=" << f_n_1 << '\n';
	for (unsigned n = 2; n <= k; ++n)
	{
		f_n = f_n_1 + f_n_2; // обчислили новий член послідовності
		f_n_2 = f_n_1;	 // значення f_n_1 і f_n "постаріли"
		f_n_1 = f_n;
		cout << "f(" << n << ")=" << f_n << '\n';
	}
	return;
}
```

У цій програмі використано так званий механізм «старе-нове» для реалізації обчислень за рекурентними формулами. Наведемо ще один приклад його використання.

Задача 11. *Задано дійсні числа a, b, ε (a > b > 0, ε > 0). Послідовності {x_i}, {y_i} утворені за правилом* $x_1 = a, y_1 = b,\ x_k = \frac{1}{2}(x_{k-1} + y_{k-1}),\ y_k = \sqrt{x_{k-1} y_{k-1}}$, *k = 2, 3, … . Знайти перше* x_n *таке, що* $|x_n - y_n| < \varepsilon$.

Для зберігання значень x_k і y_k використаємо змінні *u* і *v*, а для значень x_{k-1}, y_{k-1} – змінні *p*, *q*. Тоді для *k* = 1 значення *u* і *v* необхідно прочитати, бо їх задано. Усі наступні обчислити в циклі:

```
#include <cmath>
#include <iomanip>

void AveVsGeo()
{
	cout <<
		"\n *Побудова послідовностей середніх арифметичних геометричних*\n\n";
	unsigned k = 1;
	// початкові члени послідовностей треба прочитати
	double u, v, eps;
	cout << "Введіть a, b, eps: "; cin >> u >> v >> eps;
	cout << "\n  k      x_k         y_k\n---------------------------\n";
	cout << std::showpoint << std::fixed << '\n'
		<< std::setw(3) << k
		<< std::setw(12) << std::setprecision(5) << u
		<< std::setw(12) << std::setprecision(5) << v << '\n';
	while (std::fabs(u-v) >= eps)
	{	++k;
		// "старіння" значень
		double p = u;
		double q = v;
```

```
            // обчислення нових
            u = (p + q) / 2.0;
            v = std::sqrt(p * q);
            cout << std::setw(3) << k
                  << std::setw(12) << std::setprecision(5) << u
                  << std::setw(12) << std::setprecision(5) << v << '\n';
      }
      return;
}
```

Змінну k в цій програмі ми використали не як параметр циклу, а для обчислення номера члена послідовності $\{x_i\}$, для якого виконується умова задачі, і з метою унаочнення виведення результату. Для позначення x_k і y_k можна було використати імена x_k та y_k, проте ми хотіли продемонструвати, що й u та v працюватимуть належним чином.