ПРИКЛАДНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК,
БЕЗПЕКИ ТА МАТЕМАТИКИ, 2023, № 1

УДК 004

ОСОБЛИВОСТІ ПРОГРАМУВАННЯ МОВОЮ С++ ОЛІМПІАДНИХ ЗАДАЧ НА ВИВЕДЕННЯ

Глинчук Л.Я. (ORCID: 0000-0002-8943-9604)
Волинський національний університет імені Лесі Українки

PECULIARITIES OF PROGRAMMING IN THE C++ LANGUAGE OLYMPIAD PROBLEMS FOR DERIVATION

Hlynchuk L. Ya.
Lesya Ukrainka Volyn National University

Abstract. A few olympiad tasks are process considered and analysed on a leadingout. The selection of tasks came true on the basis of own experience of author. Tasks are examined in order of complication, id est from outages to more difficult. These tasks interesting to those that differ from ordinary complication of implementation and sometimes spend on the construction of program logic. A research format is observed following: problem specification, that contains entrance and initial data, analysis of initial conditions and analysis of algorithm for implementation in programming language, and finally, self programming of algorithm in language of C++. As, all tasks can be executed using branching and cycles only, then before programming of task the analysis of terms is executed that needs to be checked in branching. The authorial method of untiing of such tasks is shown. A programmatic code over of decisions is brought by the language of C for every task. Implementation and analysis of such tasks will give an opportunity to deepen and fasten knowledge and ability from programming.

Keywords: C++ programming language, olympiad task, cycle, branching, data output.

Свою назву олімпіадні задачі отримали від популярних змагань школярів та студентів. Олімпіадні задачі відрізняються від звичайних складністю виконання та часом затраченим на побудову логіки програми. Вони дозволяють школярам та студентам спробувати свої вміння використовувати творчий та несподіваний підхід, вивчити проблему з різних сторін. Особливістю таких задач є ще і вміння реалізувати виконання програми за заданий час та вміння затратити вказані ресурси на об'єм програми.

Олімпіади завжди були і залишаються цікавим розділом з метою виконання нестандартних задач, а їх зовнішня простота завжди оманлива та стає зрозуміла, коли переходиш до виконання.

Розбір задач етапів Всеукраїнської олімпіади з програмування можна також подивитися у відеороликах на каналах у https://www.youtube.com.

Метою роботи є демонстрація виконання підібраних задач на виведення мовою програмування с++, їх детальний розбір та аналіз.

Для реалізації задач була вибрана мова програмування С++, оскільки на даний момент детально вивчається зі студентами.

Задачі для виконання формулюються таким чином, щоб чітко було зрозуміло:

- умову;
- вхідні дані, інколи задають межі для вхідних даних і ці межі строго потрібно перевіряти при програмуванні завдання;
- вихідні дані, тобто які дані потрібно вивести як результат.

Як правило, у таких завданнях присутній приклад чи рисунок для того, аби виконавець розумів, як має виглядати результат про який описано у вихідних даних.

Розглянемо деякі задачі з описом, про який було сказано вище.

Задача 1. Для заданого натурального числа *n* вивести квадратну рамку розміром *n x n* із зірочок, заповнену проміжком як показано у прикладі на рисунку 1. Вхідні дані:

одне натуральне число *n (n ≤ 100).* Вихідні дані: виведіть прямокутну рамку розміром *n x n.* [1, ст. 72]

Input	Output
5	***** * * * * * * *****

Як бачимо, в умові задачі сказано, що на вхід буде подаватися натуральне число *n* і задана його межа *n ≤ 100.* Оскільки, натуральні числа ми використовуємо при лічбі (за означенням) то перше *n = 1* і т.д. Але при введені користувачем вхідного значення від'ємного, потрібно щоб програма або вказувала, що це помилка або нічого не робила. За логікою даного вихідного рисунку з зірочок до задачі, зрозуміло, що при вхідному значенні *n = 1* вихідне значення буде одна зірочка. При *n = 2* буде будуватися верхній і нижній рядок з зірочок. Тобто результатом буде:

```
**
**
```

При *n = 3* і т.д. уже буде будуватися квадратна рамка подібна до заданого рисунку, тобто:

```
***
* *
***
```

Мовою програмування с++ заданий на вихід рисунок-рамка – це уявна двовимірна матриця, у якій на певних позиціях стоять символи зірочка «*» та пропуск « ». Для правильного виведення цих символів потрібно організувати 2 цикли, щоб ми могли виводити символи по відповідних рядках та стовпцях. Особливістю якраз і є те, що виводити символи потрібно відповідно до виконання відповідних умов. Як бачимо, перший та останній рядок містять задану кількість зірочок, а рядки, що по середині містять зірочки тільки по краях, а решту символи то пропуски. Тому умова, яка буде перевірятися в циклах наступна: якщо ((i == 1) або (i == n) або (j == 1) або (j == n)), то виводити зірочку, в іншому випадку – пропуск, де змінні і та j відповідають за місце виведення символу.

Частина коду, що виконує виведення рамки буде мати вигляд як на рис. 1:

```
if ((n <= 100) && (n > 0))
   {for (int i = 1; i <= n; i++)
     {for (int j = 1; j <= n; j++)
       if ((i == 1) || (i == n) || (j == 1) || (j == n))
          {cout << "*";}
       else {cout << " ";}
     cout << endl;
     }
    }
```

Рис. 1. Програмний код до задачі 1

Розглянемо наступну задачу.

Задача 2. На сайті у таблиці результатів змагань, які проводяться за правилами ACM (Association for Computing Machinery), вірно розв'язана задачка оцінюється плюсом. Але він якийсь маленький. Виведіть великий плюс з зірочок. Вхідні дані: ціле число *n (1 ≤ n ≤ 100).* Вихідні дані: виведіть відповідний великий квадратний «плюс» з точок та зірочок – дивись прилади вхідних та вихідних даних таблиця 1. [1, ст. 65]

Таблиця 1. Вхідні та вихідні дані до задачі 2.

Вхідні дані # 1	Вихідні дані # 1	Вхідні дані # 2	Вихідні дані # 2
1	. * . * * * . * .	2	. . * . . . . * . . * * * * * . . * . . . . * . .

У задачі 2 теж є межі на введення вхідного даного. Тому при програмуванні це строго потрібно вказати, якщо вхідне дане виходить за межі то програма не повинна нічого робити. З таблиці до завдання видно, що великий квадратний «плюс», який потрібно отримати в результаті, повинен містити при *n = 1* – 3*3 елементів (точок та зірочок), при *n = 2* – 5*5 елементів і т.д., тобто при *n* повинно бути *(n*2 + 1)* (n*2 + 1)* елементів.

В даному випадку квадратний «плюс», аналогічно до задачі 1 теж уявна двовимірна матриця, у якій на певних позиціях стоять символи зірочка «*» та пропуск «.». Для правильного виведення цих символів потрібно організувати 2 цикли, щоб ми могли виводити відповідні символи по рядках та стовпцях. Проаналізуємо виведення по рядках. У першому рядку спочатку потрібно вивести *n* точок, тоді 1 зірочку, далі знов *n* точок. Таких однакових рядків буде *n.* В наступному рядку буде виведено *(n*2 + 1)* зірочок. А далі знову *n* попередніх рядків. Основною умовою, яку потрібно перевірити в циклах буде: якщо (i == n) або (j == n) то виводиться зірочка, інакше виводиться точка, а змінні i та j відповідають за місце виведення символу.

Програмна реалізація такого фрагменту рис. 2:

```
if ((n <= 100) && (n >= 1))
   {for (int i=0; i < (n*2 + 1); i++)
     {for (int j=0; j < (n*2 + 1); j++)
            {if ((i == n) || (j == n))
              {cout << "*";}
            else
              {cout << ".";}
             };
        cout<<endl;
     };
   }
```

Рис. 2. Програмний код до задачі 2

Розглянемо ще декілька аналогічних прикладів задач.

Задача 3. За заданим натуральним числом *n* вивести зображення розміром *n*n,* утворене символами зірочка та проміжок як показано у прикладі (рисунок 3.) Вхідні дані: одне натуральне число *n*. Вихідні дані: вивести зображення *n*n.* [1, ст. 73]

*		*		*
	*		*	
*		*		*
	*		*	
*		*		*

Рис.3. Вихідне зображення до задачі 3

Аналогічно попереднім задачам тут теж вхідне дане натуральне число, але без верхньої межі, тобто *n=1, 2, 3* і т.д. Для того, аби отримати вихідні дані у вигляді рисунку 3, потрібно проаналізувати на яких місцях стоять зірочки та скільки їх для відповідного *n.* В умові задачі сказано, що зображення повинне бути розміром *n*n.* Робимо висновок, що рисунок 3 для *n=5*, бо клітинок 5*5.

Якщо уважно подивитися, то знову маємо уявну двовимірну матрицю, у якій на певних позиціях стоять символи зірочка «*» та пропуск « ». Аналогічно двом попереднім задачам і тут потрібно використати цикли для виведення відповідних символів та перевірку умови в яких місцях виводити відповідний символ. Згідно рисунку для *n=5* у першому рядку символ «*» стоїть на позиціях 1, 3, 5; у другому рядку на 7, 9 і т.д. Побудуємо аналогічно зображення для *n=4*:

*		*	
	*		*
*		*	
	*		*

Рис. 4. Вихідне зображення до задачі 3 для *n=4*

В даному випадку, у першому рядку символ «*» стоїть на позиціях 1, 3; у другому рядку на 6, 8 і т.д. Прослідкувати таким чином не вдається, бо в першому випадку позиції зірочок непарні, в другому (при парному *n*) випадку позиції інші.

Проаналізуємо позицію символу зірочка використовуючи індекси і та j, відповідно і буде відповідати за номер рядка, а j за номер стовпця. Побудуємо таблицю для відслідкування позиції. При першому проходженні циклу, *i=1* (номер рядка) та мінятися не буде, а *j* буде набувати значень від 1 до 5 і т.д.

Таблиця 2.

i	1	1	1	1	1
j	1	2	3	4	5
(i+j)%2	Парне	не парне	Парне	не парне	Парне
символ	*	пропуск	*	Пропуск	*
I	2	2	2	2	2
J	1	2	3	4	5
(i+j)%2	не парне	Парне	не парне	Парне	не парне
символ	Пропуск	*	пропуск	*	Пропуск

Згідно таблиці 2 при *n=5* правильно будується вихідне зображення і вдалося прослідкувати залежність позицій від номеру рядка та стовпця. Якщо забрати в таблиці 2 останній стовпець, то будуть дані для *n=4* і теж все вірно будується. Тому в програмі потрібно перевіряти таку умову: якщо (i+j)%2==0 (тобто сума парна) то виводь символ зірочку, інакше символ пропуск. Відповідний фрагмент коду буду мати вигляд, рис. 5.

Задача 4. За заданим непарним натуральним числом *n* вивести зображення розміром *n*n*, утворене символами зірочка та проміжок як показано у прикладі (рисунок 6). Вхідні дані: одне непарне натуральне число *n (n > 1)*. Вихідні дані: вивести зображення *n x n*. [1, ст. 76]

```
if(n>0)
{for(int i=1; i<=n; i++)
  {for(int j=1; j<=n; j++)
    {if((i+j)%2==0)
      {cout<<"*";}
     else
      {cout<<" ";}
     }
     cout<<endl;
  }
 }
```

Рис. 5. Фрагмент коду до задачі 3

		*		
	*	*	*	
*	*	*	*	*
	*	*	*	
		*		

Рис. 6. Вихідне зображення до задачі 4

Аналогічно аналізу та розбору попередніх задач, по умові цієї задачі теж зразу можна зробити висновок, що потрібно знайти умову чи умови за допомогою яких буде відбиратися позиція виведення символу зірочка чи символу пропуск. Звернемо увагу на те, що вхідне дане непарне натуральне число і в завданні сказано, що $n > 1$ і може далі набувати значень 3, 5, 7, 9 і т.д. Рисунок 6 показує вихідне зображення n x n при $n=5$, попереднє $n=3$, а вихідне зображення буде мати вигляд:

	*	
*	*	*
	*	

Рис. 7. Вихідне зображення до задачі 4 при $n=3$

З рисунку 7 бачимо, що у першому рядку спочатку виводиться символ пропуск, тоді по середині зірочка, а далі знов пропуск. Наступний рядок всі зірочки і низ, аналогічний верху. Якщо подивитися на рисунок 6, то верх та низ симетричні відносно середнього рядка. Тому побудову такого ромба з зірочок можна розбити на 2 частини: до середнього рядка зверху та від середнього рядка донизу. Тому умови, які потрібно перевірити:

- якщо (center – i <= j && j <= center + i) то виводь зірочку, інакше пропуск, тут відходимо від центру, де поставлено першу зірочку і далі по обидва боки (верх зображення);
- якщо (center + i - n + 1 <= j && j <= center - i + n - 1) то виводь зірочку, інакше пропуск, по обидва боки від центру будуємо низ зображення.

Де center – середина умовної двовимірної матриці, n – вхідне непарне натуральне число, і та j – біжучі позиції для виведення необхідних символів. Частина програми, де описано виведення символів буде виглядати як на рисунку 8.

```
if ((n >= 1) && (n % 2!= 0))
 {int center = n / 2;
    for (i = 0; i < n; i++)
     {for (j = 0; j < n; j++)
       {if (i <= center)
          { // верхня частина ромба
          if (center - i <= j && j <= center + i)
             cout << "*";
          else cout << " ";
          }
        else { // нижня частина ромба
             if (center + i - n + 1 <= j && j <= center - i + n - 1)
                cout << "*";
             else cout << " ";
            }
        }
      cout << endl;
     }
 }
```

Рис. 8. Фрагмент коду до задачі 4

Наступна задача подібна до попередніх, бо так само потрібно вивести зображення з зірочок та проміжків (пропусків), але зображення дуже відрізняється від попередніх. Тому умови вибору позицій будуть дещо складніші.

Задача 5. За заданим натуральним числом *n* вивести зображення розміром *n x n*, утворене символами зірочка та проміжок як показано у прикладі (рисунок 9). Вхідні дані: одне натуральне число *n*. Вихідні дані: вивести зображення *n x n*. [1, ст. 73]

*	*	*	*	*
				*
*	*	*	*	*
*				
*	*	*	*	*

Рис. 9. Вихідне зображення до задачі 5

Рис. 9 показує вихідне зображення *n x n* при *n=5*. Вихідне зображення при *n=2* та *n=3* буде мати вигляд:

*	*
	*

*	*	*
		*
*	*	*

Рис. 10. Вихідне зображення до задачі 5 при *n=2* та *n=3*

*	*	*	*	*	*	*
						*
*	*	*	*	*	*	*
*						
*	*	*	*	*	*	*
						*
*	*	*	*	*	*	*

Рис. 11. Вихідне зображення до задачі 5 при *n=7*

З рисунків 9, 10 та 11 можна зробити висновок, що при збільшенні *n* зображення збільшується так як на рисунку 9 та 11, бо рисунок 10 демонструє частину зображення. При виведенні зірочок згідно рисунків можна помітити, що вони є у непарних рядках повністю та по одній по боках у решти рядках у вигляді «змійки». Маємо 3 різні рядки, на які потрібно звернути увагу і які будуть повторюватися: рядок суцільні зірочки, рядок, де одна зірочка справа та рядок, де одна зірочка зліва. Згідно цього потрібно підібрати умови для перевірки. Якщо починати нумерацію рядків та стовпців уявної двовимірної матриці з 0, то перший рядок це 0, третій рядок під номером 2 і т.д. Тобто якщо рядок парний то виводимо *n* зірочок. Далі зірочки по одній справа у рядках під номером 1, 5, 9, 13 і т.д. тобто через кожні 4 рядки; зірочки ро одній зліва у рядках під номером 3, 7, 11 і т.д., знов через кожні інші 4 рядки. Тобто зірочки ставляться через одне непарне число, але рахунок через 4 рядки в написанні коду не дуже допоможе. Потрібно додаткову змінну, яка буде контролювати чи рядок парний, а тоді після нього, перший раз ставити зірочку справа (тобто при j==n-1), на другий раз знову після парного ставити зірочку зліва (тобто при j==0). Враховуючи вище сказане, на рядки з зірочками потрібна умова: якщо номер рядка парний (i%2==0) то виводь зірочку. Або якщо рядок не парний то умова: (j==n-1 and k%2==0) or (j==0 and k%2==1), дозволяє перевірити індекси позиції, щоб поставити одну зірочку справа, якщо k парне, або зліва, якщо k непарне, де k – додаткова змінна, яка на початку 0, а потім при переході на непарний рядок збільшується на 1. Фрагмент коду за таким поясненням буде як на рисунку 12.

```
int k=0;
if (n>0)
{for(i=0; i<n; i++)
   {for(j=0; j<n; j++)
     if((i%2==0) or (j==n-1 and k%2==0) or (j==0 and k%2==1))
        cout << ("*");
     else
        cout << (" ");
     if(i%2==1)
       k++;
     cout << endl;
   }
}
```

Рис. 12. Фрагмент коду до задачі 5

Наступна задача містить цікаве виведення, яке не так легко зрозуміти за двома рисунками, які є вихідними зображеннями та відслідкувати взаємозв'язок.

Формулювання задачі, скоріше за все, потрібно розуміти як аналогічне попереднім, тобто за заданим n $(3 \le n \le 49)$ вивести зображення з зірочок та пропусків. Але формулювання з джерела таке:

Задача 6. Дивіться приклад …

Вхідні дані: одне ціле число n $(3 \le n \le 49)$. [1, ст. 68]

Input #1	Output #1
3	***
	***
	***

Input #2	Output #2
7	******* ** ** * * * * * * * * * * * ** ** *******

Оскільки, на вхід подається число строго в межах, то при написанні коду це слід враховувати так, як це робили і у попередніх задачах.

Як бачимо, з рисунків вихідних даних да задачі не дуже зрозуміло, як перейти від рисунка за вхідним даним *n=3* до рисунка за вхідним даним *n=7*. Спробуємо побудувати проміжні рисунки для *n=4, 5, 6*. Можна помітити, що перший та останній рядки містять символ зірочка у кількості *n*, по боках теж зірочки в кількості *n*. В середині побудованого квадрата ще не вистачає місця для діагоналей з пропусками, тому при *n=4* діагоналі закривають середину, а вихідний рисунок до задачі буде мати вигляд:

*	*	*	*
*	*	*	*
*	*	*	*
*	*	*	*

Рис. 13. Вихідний рисунок до задачі 6 при *n=4*

Побудуємо для *n=5* і спробуємо в середині відобразити діагоналі, аналогічно як для рисунку при *n=7* (рисунок 14).

*	*	*	*	*
*	*		*	*
*		*		*
*	*		*	*
*	*	*	*	*

Рис. 14. Вихідний рисунок до задачі 6 при *n=5*

І, нарешті, при *n=6*, тобто уже при парному вхідному даному вихідний рисунок буде (рисунок 15):

*	*	*	*	*	*
*	*			*	*
*		*	*		*
*		*	*		*
*	*			*	*
*	*	*	*	*	*

Рис. 15. Вихідний рисунок до задачі 6 при *n=6*

З рисунків 13, 14, 15 та рисунків умови задачі можна зробити висновок, що при парному *n>4* по середині вибудовується квадрат з 4-х зірочок, при непарному *n>3* там виводиться лише одна зірочка, тобто зірочки виводяться по діагоналях. Крім описаного виведення зірочок, не потрібно забувати про рядок зверху-знизу та справа-зліва. Тому для виведення вихідного рисунку лишилося зібрати все описане, для визначення позиції, в умови. Можна зробити 2-ма способами, спочатку вивести рядок з *n* зірочок, тоді побудувати фігуру посередині, потім знов вивести рядок з *n* зірочок. Умова в такому випадку буде: (j == 0 || j == n-1 || j == ((n-1) - i) || j == i), починаємо відлік умовних рядків та стовпців з 0, тоді перша частина умови – ліво-право, друга частина – дві діагоналі, де і та j біжучі позиції для виведення відповідних символів. Фрагмент коду буде мати вигляд:

```
if(n<=49 && n>=3)
{for (int i = 0; i < n; i++)
     {cout << "*";}
      cout << endl;
  for (int i = 1; i < n-1; i++)
     {for (int j = 0; j < n; j++)
          {if (j == 0 || j == n-1 || j == ((n-1) - i) ||j == i )
              {cout << "*";}
           else
              {cout << " ";}
          }
     cout << endl;
     }
  for (int i = 0; i < n; i++)
    {cout << "*";}
 cout << endl;
 }
```

Рис. 16. Фрагмент коду до задачі 6 (перший спосіб)

Другий спосіб виведення заданої фігури з символів, полягає в пошуку позиції для зірочки та пропуску згідно умови, яка включає верхній та нижній рядки. Така загальна умова буде мати вигляд: (i == 0 || i == n-1 || j == 0 || j == n-1 || j == n-1 – i || i == j), тут перші 2 умови, то верхній та нижній рядок, наступні 2 умови, то правий та лівий стовпці, решту 2 умови – обидві діагоналі. Фрагмент коду буде як на рисунку 17.

```
int n=11;
for(int i=0; i<n; i++)
   {for(int j=0; j<n; j++)
     if(i == 0 || i == n-1 || i == j || j == 0 || j == n-1 || j == n-1 - i)
       {cout << "*";}
     else
       {cout << " ";}
     cout << endl;
   }
```

Рис. 17. Фрагмент коду до задачі 6 (другий спосіб)

Наступна задача, просить вивести інші символи, але подібна до попередніх виконанням.

Задача 7. Вася хоче надрукувати на принтері піраміду з якогось символу висотою *h*. Напишіть програму, яка допоможе йому в цьому, не забуваючи, що програма повинна бути «економічно вигідною» тобто друкувати найменшу кількість символів.

Приклади пірамід наведено у прикладах вхідних та вихідних даних. [2]

Вхідні дані #1 Вихідні дані #1
A 3 12

```
  A
 AAA
AAAAA
```

Вхідні дані #2 Вихідні дані #2
M 9 117

```
    M
   MMM
  MMMMM
 MMMMMMM
MMMMMMMMM
```

Для цієї задачі потрібно виводити заданий символ та, хоч і в завданні не сказано, але до початку виведення першого символу і в наступних рядках потрібно виводити ще і символ пропуск. «Економічно вигідна» побудова і полягає в тому, щоб вивести найменшу кількість символів разом: пропуск +заданий символ. З перших вихідних даних видно, що літер є 9, а пропусків 3 (у першому рядку 2 та у другому 1), згідно другого прикладу можна порахувати, що пропусків є 1+2+3+4+5+6+7+8=36, літер 1+3+5+7+9+11+13+15+17=81, разом 117 як і у вихідних даних. Для того, аби вивести кількість символів ми можемо або рахувати їх під час виведення, або вказати формулу, яка буде залежати від висоти піраміди. За підрахунками вище, помітно, що кількість літер рівна *h*h* (9, 81). А кількість пропусків як половина площі прямокутника, тобто *h*(h-1)*/2 (3, 36). Разом кількість усіх символів буде обчислюватися формулою *h*h+ h*(h-1)*/2.

Оскільки, задана висота і кількість рядків повинна бути рівна висоті, то перший цикл буде від 1 до *h*. У першому рядку спочатку потрібно вивести *h-1* пропуск, у наступному рядку *h-2* пропуски і т.д. Тому другий цикл буде від 0 до *h-i*, де і відповідає номеру рядка. Після пропусків потрібно вивести літеру, у першому рядку 1 літеру, у другому 3, у третьому 5 і т.д., то другий цикл буде від 0 до *2*i -1*. Після виведення рядка пропусків та літер потрібно перейти на новий рядок. Весь фрагмент коду буде як на рисунку 18.

```
cout << h*h + h*(h - 1)/2 << endl;
for (i = 1; i <= h; i++)
  { for (j = 0; j < h - i; j++)
        cout << " ";
      for (k = 0; k < 2 * i - 1; k++)
        cout << a;
    cout << endl;
}
```

Рис. 18. Фрагмент коду до задачі 7

У всіх випадках виконання задач потрібно звертати увагу, на заданий об'єм програми та на заданий час роботи програми. В олімпіадних задачах це важлива умова, якщо її не дотримуватися, то завдання можуть бути не зараховані. Якщо ви розумієте,

що є декілька способів виконання, то мусите обрати той варіант виконання, який задовольняє і час і об'єм.

Висновки. Шаблонів задач на виведення вистачає, для того, аби навчитися будувати правильні умови в конструкції розгалуження. У роботі було відібрано оригінальні та цікаві задачі для того, аби показати різноманітність умов, які потрібно перевіряти для знаходження позиції для виведення символів. Як бачимо, для розв'язування наведених задач, кожен програмний код містить тільки вкладені цикли for та розгалуження if (перевірки умов). Можна сказати, що виконання таких задач можливе після вивчення тем розгалуження та вкладені цикли для поглиблення та закріплення знань та вмінь програмувати та застосовувати дані розділи на практиці.

Бібліографія

1. Жмурко О. І. Олімпіади з програмування. Прості задачі / О. І. Жмурко, Т. О. Охріменко; МОН України, Уманський держ. пед. ун-т імені Павла Тичини. Умань: Візаві, 2020. 298 с.
2. Піраміда з символів. URL: https://www.eolymp.com/uk/problems/1119
3. Методичні рекомендації щодо розв`язання олімпіадних задач з програмування. URL: http://surl.li/fmiei
4. Розбір задач ІІ етапу Всеукраїнської олімпіади з програмування. URL: http://surl.li/fmiiw

References

1. Zhmurko O. I. Programming Olympics. Simple problems / O. I. Zhmurko, T. O. Okhrimenko; Ministry of Education, Culture, Sports, Science and Technology of Ukraine, Uman State Pavlo Tychyna University. Uman: Visavy, 2020. 298 p.
2. Pyramid of symbols. URL: https://www.eolymp.com/uk/problems/1119
3. Methodical recommendations for solving Olympiad programming problems. URL: http://surl.li/fmiei
4. Analysis of tasks of the II stage of the All-Ukrainian Programming Olympiad. URL: http://surl.li/fmiiw