**Лабораторна робота №5. Циклічні конструкції**

1. Вимоги
   1. Розробник

* Бельчинська Катерина Юріївна
* студентка групи КІТ-320
* 04-nov-2020
  1. Індивідуальне завдання

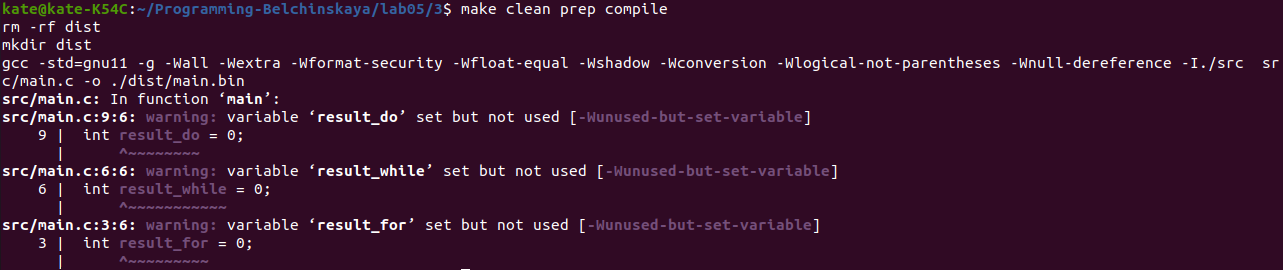
Визначити, чи є задане 6-значне число “щасливим” квитком (“щасливий квиток” – квиток, в якому сума першої половини чисел номера дорівнює сумі другої половини. Наприклад, білет з номером 102300 є щасливим, бо 1+0+2=3+0+0). Реалізувати програму за допомогою трьох циклів: for, while-do, do-while.

1. Опис роботи

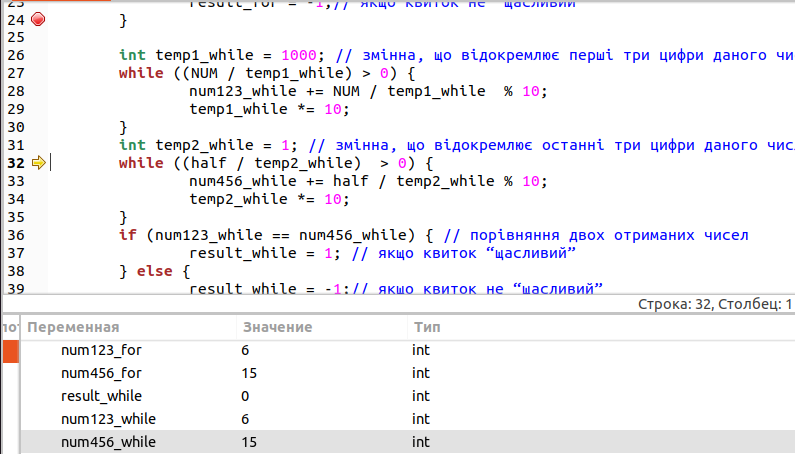
Код програми:

|  |
| --- |
| #define NUM 123456 |
|  | int main () { |
|  | int result\_for = 0; |
|  | int num123\_for = 0; |
|  | int num456\_for = 0; |
|  | int result\_while = 0; |
|  | int num123\_while = 0; |
|  | int num456\_while = 0; |
|  | int result\_do = 0; |
|  | int num123\_do = 0; |
|  | int num456\_do = 0; |
|  | int half = NUM % 1000; |
|  | for (int i = 1000; (NUM / i) > 0; i \*= 10) { /\* цикл відокремлює перші три цифри даного числа та сумує їх, знаходимо перше число “щасливого” квитку \*/ |
|  | num123\_for += NUM / i % 10; |
|  | } |
|  | for (int j = 1; (half / j) > 0; j \*= 10) { /\* цикл відокремлює останні три цифри даного числа та сумує їх, знаходимо друге число “щасливого” квитку\*/ |
|  |  |
|  | num456\_for += half / j % 10; |
|  | } |
|  | if (num123\_for == num456\_for) { // порівняння двох отриманих чисел |
|  | result\_for = 1; // якщо квиток “щасливий” |
|  | } else { |
|  | result\_for = -1;// якщо квиток не “щасливий” |
|  | } |
|  |  |
|  | int temp1\_while = 1000; // змінна, що відокремлює перші три цифри даного числа, щоб сумувати їх у циклі |
|  | while ((NUM / temp1\_while) > 0) { |
|  | num123\_while += NUM / temp1\_while % 10; |
|  | temp1\_while \*= 10; |
|  | } |
|  | int temp2\_while = 1; // змінна, що відокремлює останні три цифри даного числа, щоб сумувати їх у циклі |
|  | while ((half / temp2\_while) > 0) { |
|  | num456\_while += half / temp2\_while % 10; |
|  | temp2\_while \*= 10; |
|  | } |
|  | if (num123\_while == num456\_while) { // порівняння двох отриманих чисел |
|  | result\_while = 1; // якщо квиток “щасливий” |
|  | } else { |
|  | result\_while = -1;// якщо квиток не “щасливий” |
|  | } |
|  |  |
|  |  |
|  | int temp1\_do = 1000; // змінна, що відокремлює перші три цифри даного числа, щоб сумувати їх у циклі |
|  | do { |
|  | num123\_do += NUM / temp1\_do % 10; |
|  | temp1\_do \*= 10; |
|  | } while ((NUM / temp1\_do) > 0); |
|  | int temp2\_do = 1; // змінна, що відокремлює останні три цифри даного числа, щоб сумувати їх у цикл |
|  | do { |
|  | num456\_do += half / temp2\_do % 10; |
|  | temp2\_do \*= 10; |
|  | } while ((half / temp2\_do) > 0); |
|  | if (num123\_do == num456\_do) { // порівняння двох отриманих чисел |
|  | result\_do = 1; // якщо квиток “щасливий” |
|  | } else { |
|  | result\_do = -1;// якщо квиток не “щасливий” |
|  | } |
|  | return 0; |
|  | } |

Компіляція програми :



Відладка програми:



1. Варіанти використання.

Програму доцільно використовувати для визначення “щасливості” квитка.

Висновок

Отже, визначено “щасливий” квиток за допомогою трьох типів циклів: for, while do, do while.

1. Загальне завдання

1

|  |
| --- |
| #define X 16// задане число |
|  | #define Y 20// задане число |
|  | int main () { |
|  | int num1; |
|  | int num2; |
|  | int nod\_for = 0; |
|  | int nod\_while = 0; |
|  | int nod\_do = 0; |
|  | for (num1 = X, num2 = Y; num1 > 0 && num2 > 0;) {/\*ділимо більше число на менше доки остача від ділення не буде 0 (алгоритм Евкліда)\*/ |
|  |  |
|  | (num1 > num2) ? (num1 %= num2) : (num2 %= num1); |
|  | nod\_for = num1 + num2; |
|  | } |
|  | num1 = X;//щоб увійти у новий цикл, минулі значення змінних необхідно "скинути" |
|  | num2 = Y; |
|  | while (num1 > 0 && num2 > 0) { /\*ділимо більше число на менше доки остача від ділення не буде 0\*/ |
|  | if (num1 > num2) { |
|  | num1 %= num2; |
|  | } else if (num1 < num2) { |
|  | num2 %= num1; |
|  | nod\_while = num1 + num2; |
|  | } |
|  | } |
|  | num1 = X; |
|  | num2 = Y; |
|  | do { /\*ділимо більше число на менше доки остача від ділення не буде 0\*/ |
|  |  |
|  | if (num1 > num2) { |
|  | num1 %= num2; |
|  | } else if (num1 < num2) { |
|  | num2 %= num1; |
|  | } |
|  | nod\_do = num1 + num2; |
|  | } while (num1 > 0 && num2 > 0); |
|  | return 0; |
|  | } |

2

|  |
| --- |
| #define NUM 3//задане число |
|  | int main () { |
|  | int tempFor = 0; |
|  | int resultFor = 0; |
|  | for (int i = 2; i < NUM; i++) {/\*якщо число ділиться націло (тобто остача від ділення =0) на число менше за нього, то воно складене\*/ |
|  | tempFor = NUM % i; |
|  | if (tempFor == 0) { |
|  | resultFor = 0;//число складене |
|  | } else { |
|  | resultFor = 1;//число просте |
|  | } |
|  | } |
|  | int i = 2; |
|  | int resultWhile = 0; |
|  | int tempWhile = 0; |
|  | while (i < NUM) { |
|  | tempWhile = NUM % i; |
|  | if (tempWhile == 0) { |
|  | resultWhile = 0;//число складене |
|  | } else { |
|  | resultWhile = 1;//число просте |
|  | } |
|  | i++; |
|  | } |
|  | int j = 2; |
|  | int resultDo = 0; |
|  | int tempDo = 0; |
|  | do { |
|  | tempDo = NUM % j; |
|  | if (tempDo == 0) { |
|  | resultDo = 0;//число складене |
|  | } else { |
|  | resultDo = 1;//число просте |
|  | } |
|  | j++; |
|  | } while (j < NUM); |
|  | return 0; |
|  | } |

4

|  |
| --- |
| #define NUM 6 //задане число |
|  | int main () { |
|  | int sum = 0; |
|  | int temp\_for = 0; |
|  | int result\_for = 0; |
|  | for (int i = 1; i < NUM; i++) {//ділимо дане число націло (тобто остача = 0) |
|  | temp\_for = NUM % i; |
|  | if (temp\_for == 0) {//якщо остача від ділення 0 - сумуємо дільники |
|  | sum += i; |
|  | } |
|  | } |
|  | if (sum == NUM) { |
|  | result\_for = 1;//число досконале |
|  | } |
|  | else { |
|  | result\_for = 0;//число не досконале |
|  | } |
|  | int sum\_while = 0; |
|  | int temp\_while = 0; |
|  | int result\_while = 0; |
|  | int step\_while = 1;//ділимо дане число націло |
|  | while (step\_while < NUM) { |
|  | temp\_while = NUM % step\_while; |
|  | if (temp\_while == 0) {//якщо остача від ділення 0 - сумуємо дільники |
|  | sum\_while += step\_while; |
|  | } |
|  | step\_while++; |
|  | } |
|  | int sum\_do = 0; |
|  | int temp\_do = 0; |
|  | int result\_do = 0; |
|  | int step\_do = 1; |
|  | do { |
|  | temp\_do = NUM % step\_do; |
|  | if (temp\_do == 0) { |
|  | sum\_do += step\_do; |
|  | } |
|  | step\_do++; |
|  | } while (step\_do < NUM); |
|  | return 0; |
|  | } |

5

|  |
| --- |
| #define X 25 |
|  | int main () { |
|  | int num1 = X; |
|  | int sqrtFor = 0; |
|  | int sqrtWhile = 0; |
|  | int sqrtDo = 0; |
|  | int tempWhile = 1; |
|  | int tempDo = 1; |
|  | for (int tempFor = 1; tempFor< num1; tempFor++) { |
|  | if (tempFor == (tempFor + (num1 / tempFor)) / 2) { |
|  | sqrtFor = tempFor; |
|  | } |
|  | } |
|  | num1 = X; |
|  | while (tempWhile < num1) { |
|  | if (tempWhile == (tempWhile + (num1 / tempWhile)) / 2) { |
|  | sqrtWhile = tempWhile; |
|  | } |
|  | tempWhile++; |
|  | } |
|  | num1 = X; |
|  | do { |
|  | if (tempDo == (tempDo + (num1 / tempDo)) / 2) { |
|  | sqrtDo = tempDo; |
|  | } |
|  | tempDo++; |
|  | } while (tempDo < num1); |
|  | return 0; |
|  | } |