

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** 2

з дисципліни “ Основи програмування ”

тема “ Циклічні конструкції та робота з перечисленнями і статичними масивами даних”

|  |  |  |
| --- | --- | --- |
| Виконав  студент I курсу  групи КП-52  Саприкін Артем Олексійович  (*прізвище, ім’я, по батькові*)  варіант № 14 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Штрафні бали:   |  |  | | --- | --- | | **Термін здачі** | **Оформлення звіту** | |  |  | | Нараховані бали:   |  |  |  | | --- | --- | --- | | **Корект. програм (2 бала)** | **Відп. на теор. питання (1 бал)** | **Відп. на прогр. питання (2 бала)** | |  |  |  | | Сумарний бал:   |  | | --- | |  | |

Київ 2014

**Мета роботи**

Навчитися працювати зі статичними масивами різних типів даних мови програмування С. Застосувати на практиці різні види циклічних конструкцій для обчислення математичних формул і при роботі з одномірними та багатомірними масивами даних. Застосувати перечислення при вирішенні різнопланових завдань.

**Постановка завдання**

**Завдання 1.** За допомогою ітераційних циклічних конструкцій обчислити значення виразу:

**Завдання 2.** Реалізувати функції для роботи з одномірними масивами

|  |
| --- |
| Список функцій |
| /\* Заповнює масив випадковими цілими числами в діапазоні [-255..255] (включно) \*/  void fillRand2(int arr[], int size);  /\* Перевіряє чи всі елементи масиву знаходяться у діапазоні [-255..255] (включно). Повертає 1, якщо умова задовольняється і 0 - якщо не задовольняється \*/  int checkRand2(int arr[], int size);  /\* Знаходить максимальний серед елементів масиву \*/  int maxValue(int arr[], int size);  /\* Знаходить індекс першого максимального елемента масиву \*/  int maxIndex(int arr[], int size);  /\* Знаходить значення, яке найчастіше зустрічається у масиві. Якщо таких декілька, повертає більше з них\*/  int maxOccurance(int arr[], int size);  /\* Розміри arr1, arr2 і res одинакові. Перевіряється різниця між відповідними елементами масивів arr1 і arr2. Ця різниця записується у відповідний елемент масиву res. Якщо всі різниці рівні 0 (тобто масиви arr1 і arr2 одинакові, то функція повертає 1, інакше - 0). \*/  int diff(int arr1[], int arr2[], int res[], int size);  /\* сума \*/  void add(int arr1[], int arr2[], int res[], int size);  /\* lt - less than \*/  int lt(int arr1[], int arr2[], int size);  /\* lor - logical OR \*/  void lor(int arr1[], int arr2[], int res[], int size); |

**Завдання 3.**  Таблицею описується кінцевий автомат

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **States\Input** | **6** | **14** | **24** | **204** |
| **0** | **9**, 1 | **break** | **7**, 2 | **5**,3 |
| **1** | **3**, 2 |  | **1,** 2 | **pop**, 0 |
| **2** | **repeat**, 0 | **break** | **continue,** 0 |  |
| **3** | **19,** 0 | **pop,** 0 |  | **14,** 1 |

У першому стовпці знаходяться числові назви станів автомата.

На початку роботи автомат знаходиться у стані **0**.

У першому рядку знаходяться всі можливі вхідні символи (вхідний алфавіт автомата). Вхідні символи отримуються з масиву **moves**.

Автомат послідовно зчитує елементи з вхідного масиву **moves**.Коли весь вхідний масив був зчитаний - робота автомату зупиняється.

У клітинках, на перетині стану і символу подаються **переходи**, як пара (операція, новий стан), розділена комою:

* Перше значення - **операція**
* Друге значення - **новий стан**, у який перейде масив після виконання операції.

Операції:

* **push** - записана у вигляді числа, яке потрібно записати у вихідний масив **res**. Результуючий масив має обмежений розмір, тому коли автомат заповнить всі елементи результату, роботу потрібно завершити.
* **pop** - із вихідної послідовності видалиться останній доданий елемент. Якщо видалено
* вже всі елементи, а поточна операція **pop** - то завершити роботу автомата
* **continue -** пуста операція, нічого не відбувається.
* **repeat -** пуста операція, але вхідне значення використається у наступній ітерації.
* **break -** операція, що зупиняє роботу автомату

.

Пусті клітинки означають **недозволені переходи**. Якщо на якійсь ітерації потрібно зробити недозволений перехід, то робота автомату припиняється.

Якщо на вхід автомата попало **число, що не описане у таблиці**, то автомат зупиняється.

Всі вільні комірки вихідної послідовності **заповнити нулями**.

Функція повинна повертати **кількість заповнених елементів** у результуючому масиві (без врахування нульових).

**При реалізації автомата обов’язково використати перечислення (enum) і двомірні масиви.**

**Тексти коду програм**

**Завдання 1.**

|  |
| --- |
| loops.c |
| #include <stdio.h>  #include <math.h>  const int n0 = 4;  const int n1 = 1;  double calc (int n, int m)  {  double x = 0.;  for (int i = 1; i <= n; i++)  for (int j = 1; j <= m; j++)  x += cos (i\*M\_PI/2) + (n1 + 1)\*sin(j\*M\_PI/4);  return x;  } |

**Завдання 2.**

|  |
| --- |
| arrays.c |
| #include <stdio.h>  #include <math.h>  #include <stdlib.h>  const int n0 = 4;  const int n1 = 1;  const int nMax = 511;  int counts[nMax];  int max (int x, int y)  {  return (x > y) ? x : y;  }  int min (int x, int y)  {  return x < y ? x : y;  }  void fillRand2 (int arr[], int size)  {  for (int i = 0; i < size; i++)  arr[i] = (rand () % 511) - 255;  }  int checkRand2 (int arr[], int size)  {  for (int i = 0; i < size; i++)  if (arr[i] > 255 && arr[i] < -255)  return 0;  return 1;  }  int maxValue (int arr[], int size)  {  int m = arr[0];  for (int i = 0; i < size; i++)  m = max (m, arr[i]);  return m;  }  int minIndex (int arr[], int size)  {  int m = arr[0], index = -1;  for (int i = 0; i < size; i++)  m = min (m, arr[i]);  for (int i = 0; i < size; i++)  if (arr[i] == m)  {  index = i;  break;  }  return index;  }  int maxOccurance (int arr[], int size)  {  int m = 0;  for (int i = 0; i < size; i++)  m = max (m, ++counts[arr[i] + 255]);  for (int i = nMax - 1; i >= 0; i--)  if ( counts[i] == m )  return i - 255;  }  int diff (int arr1[], int arr2[], int res[], int size)  {  int bZero = 1;  for (int i = 0; i < size; i++)  {  res[i] = arr1[i] - arr2[i];  if (res[i] != 0)  bZero = 0;  }  }  void add (int arr1[], int arr2[], int res[], int size)  {  for (int i = 0; i < size; i++)  res[i] = arr1[i] + arr2[i];  }  int gteq (int arr1[], int arr2[], int size)  {  for (int i = 0; i < size; i++)  if (arr1[i] < arr2[i])  return 0;  return 1;  }  void lor (int arr1[], int arr2[], int res[], int size)  {  for (int i = 0; i < size; i++)  res[i] = arr1[i] || arr2[i];  }  void printArr (int arr[], int size)  {  for (int i = 0; i < size; i++)  printf ("%d ", arr[i]);  printf ("\n\n");  } |

**Завдання 3.**

|  |
| --- |
| automata.c |
| #include <stdio.h>  #include <stdlib.h>  enum Move {POP = -4, REPEAT = -3, CONTINUE = -2, BREAK = -1};  struct Pair  {  int op;  int state;  };  struct Pair Table[4][210];  void fillTable ()  {  struct Pair p;  p.op = 9;  p.state = 1;  Table[0][6] = p;  p.op = BREAK;  p.state = -1;  Table[0][14] = p;  p.op = 7;  p.state = 2;  Table[0][24] = p;  p.op = 5;  p.state = 3;  Table[0][204] = p;  p.op = 3;  p.state = 2;  Table[1][6] = p;  p.op = BREAK;  p.state = -1;  Table[1][14] = p;  p.op = 1;  p.state = 2;  Table[1][24] = p;  p.op = POP;  p.state = 0;  Table[1][204] = p;  p.op = REPEAT;  p.state = 0;  Table[2][6] = p;  p.op = BREAK;  p.state = -1;  Table[2][14] = p;  p.op = CONTINUE;  p.state = 0;  Table[2][24] = p;  p.op = BREAK;  p.state = -1;  Table[2][204] = p;  p.op = 19;  p.state = 0;  Table[3][6] = p;  p.op = POP;  p.state = 0;  Table[3][14] = p;  p.op = BREAK;  p.state = -1;  Table[3][24] = p;  p.op = 14;  p.state = 1;  Table[3][204] = p;  }  int run (int moves[], int movesLen, int res[], int resLen)  {  fillTable ();  for (int i = 0; i < resLen; i++)  res[i] = 0;  int state = 0;  int cur = 0;  for (int i = 0; i < movesLen; i++)  {  switch (Table[state][ moves[i] ].op)  {  case POP:  res[cur--] = 0;  if (cur < 0)  return 0;  state = Table[state][ moves[i] ].state;  break;  case REPEAT:  state = Table[state][ moves[i] ].state;  i--;  break;  case CONTINUE:  state = Table[state][ moves[i] ].state;  continue;  case BREAK:  return cur;  break;  default:  res[cur++] = Table[state][ moves[i] ].op;  state = Table[state][ moves[i] ].state;  if (cur >= resLen)  return cur;  break;  }  }  return cur;  } |

**Приклади результатів**

**Завдання 1.**

1. n = 5, m = 10

res = 17.07

1. n = 2, m = 20

res = ­–10.34

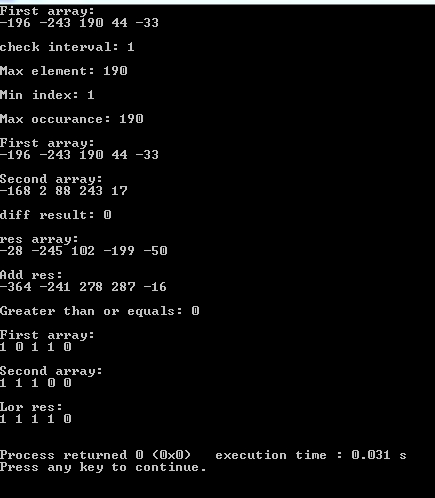
1. n = 10, m = 10

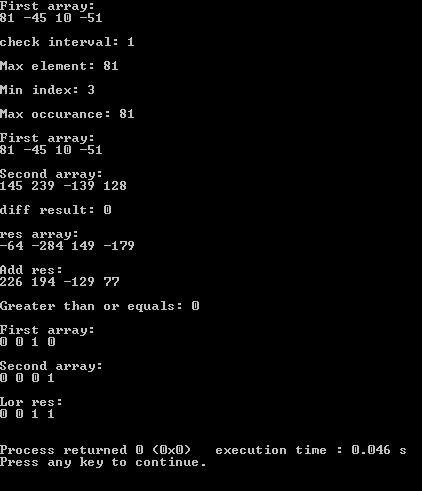
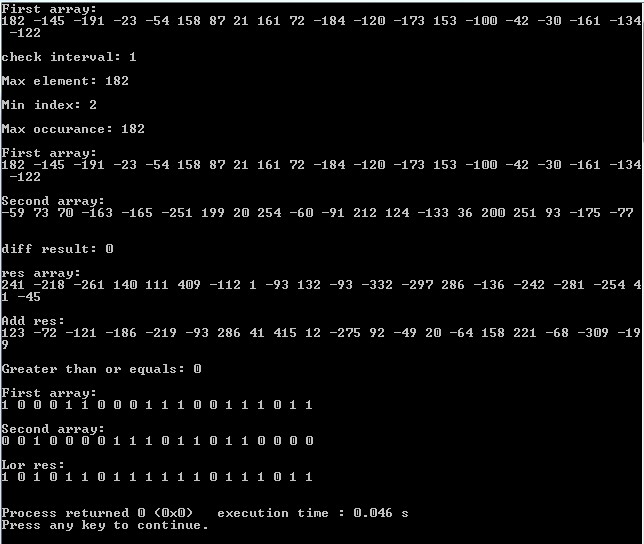
res = 24.14

1. n = 1, m = 1

res = 1.41

**Завдання 2.**



**Завдання 3.**

1. moves[] = {6, 204, 24, 6, 24, 24, 6} resLen = 10;

res = 4;

7 9 1 9 0 0 0 0 0 0

1. moves[] = {6, 204, 24, 14, 24, 24, 24} resLen = 10;

res = 1

7 0 0 0 0 0 0 0 0 0

1. moves[] = {14} resLen = 10;

res = 0

0 0 0 0 0 0 0 0 0 0

1. moves[] = {6, 204, 24, 6, 24, 24, 6} resLen = 2;

res = 2

7 9 0 0 0 0 0 0 0 0

**Висновки**

Виконавши дану лабораторну роботу, були опрацьовані статичні масиви різних типів даних мови програмування С. Також було застосовано на практиці різні види циклічних для обчислення математичних формул і при роботі одномірними та багатомірними масивами даних. В результаті завдання 3 було застосовано перечислення і двомірні масиви для демонстрації спрощення коду при використанні цих способів.

Компіляція всього коду відбувалася за допомогою компілятора gcc. Робота програм була протестована за допомогою допоміжного програмного коду, текст якого наведено у додатку А.

**Додаток А**

Тестування коду loops.c

|  |
| --- |
| test\_loops.c |
| #include <stdio.h>  #include <math.h>  const int n0 = 4;  const int n1 = 1;  double calc (int n, int m)  {  double x = 0.;  for (int i = 1; i <= n; i++)  for (int j = 1; j <= m; j++)  x += cos (i\*M\_PI/2) + (n1 + 1)\*sin(j\*M\_PI/4);  return x;  }  int main ()  {  int n = 5, m = 10;  printf ("%llf", calc (n, m));  return 0;  } |

Тестування коду завдання arrays.c

|  |
| --- |
| test\_arrays.c |
| #include <stdio.h>  #include <math.h>  #include <stdlib.h>  #include <time.h>  const int n0 = 4;  const int n1 = 1;  const int nMax = 511;  int counts[nMax];  int max (int x, int y)  {  return (x > y) ? x : y;  }  int min (int x, int y)  {  return x < y ? x : y;  }  void fillRand2 (int arr[], int size)  {  for (int i = 0; i < size; i++)  arr[i] = (rand () % 511) - 255;  }  int checkRand2 (int arr[], int size)  {  for (int i = 0; i < size; i++)  if (arr[i] > 255 && arr[i] < -255)  return 0;  return 1;  }  int maxValue (int arr[], int size)  {  int m = arr[0];  for (int i = 0; i < size; i++)  m = max (m, arr[i]);  return m;  }  int minIndex (int arr[], int size)  {  int m = arr[0], index = -1;  for (int i = 0; i < size; i++)  m = min (m, arr[i]);  for (int i = 0; i < size; i++)  if (arr[i] == m)  {  index = i;  break;  }  return index;  }  int maxOccurance (int arr[], int size)  {  int m = 0;  for (int i = 0; i < size; i++)  m = max (m, ++counts[arr[i] + 255]);  for (int i = nMax - 1; i >= 0; i--)  if ( counts[i] == m )  return i - 255;  }  int diff (int arr1[], int arr2[], int res[], int size)  {  int bZero = 1;  for (int i = 0; i < size; i++)  {  res[i] = arr1[i] - arr2[i];  if (res[i] != 0)  bZero = 0;  }  }  void add (int arr1[], int arr2[], int res[], int size)  {  for (int i = 0; i < size; i++)  res[i] = arr1[i] + arr2[i];  }  int gteq (int arr1[], int arr2[], int size)  {  for (int i = 0; i < size; i++)  if (arr1[i] < arr2[i])  return 0;  return 1;  }  void lor (int arr1[], int arr2[], int res[], int size)  {  for (int i = 0; i < size; i++)  res[i] = arr1[i] || arr2[i];  }  void printArr (int arr[], int size)  {  for (int i = 0; i < size; i++)  printf ("%d ", arr[i]);  printf ("\n\n");  }  int main ()  {  srand (time (NULL));  const int size = 2;  int arr1[size];  int arr2[size];  int res[size];  fillRand2 (arr1, size);  fillRand2 (arr2, size);  printf ("First array: \n");  printArr (arr1, size);  printf ("check interval: %d\n\n", checkRand2 (arr1, size));  printf ("Max element: %d\n\n", maxValue (arr1, size));  printf ("Min index: %d\n\n", minIndex (arr1, size));  printf ("Max occurance: %d\n\n", maxOccurance (arr1, size));  printf ("First array: \n");  printArr (arr1, size);  printf ("Second array: \n");  printArr (arr2, size);  printf ("diff result: %d\n\n", diff (arr1, arr2, res, size));  printf ("res array: \n");  printArr (res, size);  add (arr1, arr2, res, size);  printf ("Add res: \n");  printArr (res, size);  printf ("Greater than or equals: %d\n\n", gteq (arr1, arr2, size));  for (int i = 0; i < size; i++)  {  arr1[i] = (arr1[i] + 255) % 2;  arr2[i] = (arr2[i] + 255) % 2;  }  printf ("First array: \n");  printArr (arr1, size);  printf ("Second array: \n");  printArr (arr2, size);  lor (arr1, arr2, res, size);  printf ("Lor res: \n");  printArr (res, size);  return 0;  } |

Тестування коду завдання automata.c

|  |
| --- |
| test\_automata.c |
| #include <stdio.h>  #include <stdlib.h>  enum Move {POP = -4, REPEAT = -3, CONTINUE = -2, BREAK = -1};  struct Pair  {  int op;  int state;  };  struct Pair Table[4][210];  void fillTable ()  {  struct Pair p;  p.op = 9;  p.state = 1;  Table[0][6] = p;  p.op = BREAK;  p.state = -1;  Table[0][14] = p;  p.op = 7;  p.state = 2;  Table[0][24] = p;  p.op = 5;  p.state = 3;  Table[0][204] = p;  p.op = 3;  p.state = 2;  Table[1][6] = p;  p.op = BREAK;  p.state = -1;  Table[1][14] = p;  p.op = 1;  p.state = 2;  Table[1][24] = p;  p.op = POP;  p.state = 0;  Table[1][204] = p;  p.op = REPEAT;  p.state = 0;  Table[2][6] = p;  p.op = BREAK;  p.state = -1;  Table[2][14] = p;  p.op = CONTINUE;  p.state = 0;  Table[2][24] = p;  p.op = BREAK;  p.state = -1;  Table[2][204] = p;  p.op = 19;  p.state = 0;  Table[3][6] = p;  p.op = POP;  p.state = 0;  Table[3][14] = p;  p.op = BREAK;  p.state = -1;  Table[3][24] = p;  p.op = 14;  p.state = 1;  Table[3][204] = p;  }  int run (int moves[], int movesLen, int res[], int resLen)  {  fillTable ();  for (int i = 0; i < resLen; i++)  res[i] = 0;  int state = 0;  int cur = 0;  for (int i = 0; i < movesLen; i++)  {  switch (Table[state][ moves[i] ].op)  {  case POP:  res[cur--] = 0;  if (cur < 0)  return 0;  state = Table[state][ moves[i] ].state;  break;  case REPEAT:  state = Table[state][ moves[i] ].state;  i--;  break;  case CONTINUE:  state = Table[state][ moves[i] ].state;  continue;  case BREAK:  return cur;  break;  default:  res[cur++] = Table[state][ moves[i] ].op;  state = Table[state][ moves[i] ].state;  if (cur >= resLen)  return cur;  break;  }  }  return cur;  }  int main()  {  int moves[] = {6, 204, 24, 6, 24, 24, 6};  int movesLen = sizeof (moves) / sizeof (moves[0]);  int resLen = 10;  int res[resLen];  printf ("Result: %d\n", run (moves, movesLen, res, resLen));  for (int i = 0; i < resLen; i++)  printf ("%d ", res[i]);  return 0;  } |