Скобелкин Глеб ФИб-1301  
Лабораторная работа №2  
  
№1.

Теоретически эту задачу решить можно, но ресурсы системы не позволяют.

№2.

def primes(n):

c = 0

n1 = n + 1

a = [True]\*n1

for k in range(2, int(n1\*\*0.5) + 1):

if a[k]:

for i in range(k\*k, n1, k):

a[i] = False

for i in range(2, n1):

if a[i]:

c += 1

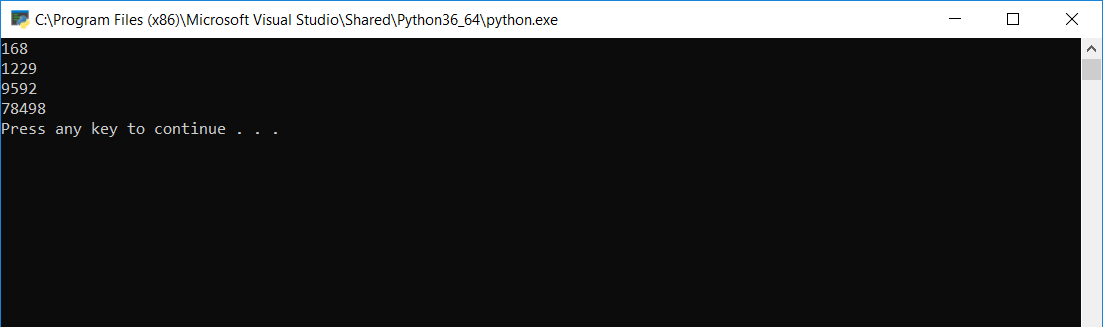
return c

print(primes(10\*\*3))

print(primes(10\*\*4))

print(primes(10\*\*5))

print(primes(10\*\*6))



№3.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| t | 1 | 1 | 3 | 1 | 3 | 5 | 7 | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 1 | 3 | 5 | 7 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| t | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 | 27 | 29 | 31 | 1 | 3 | 5 | 7 | 9 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
| t | 13 | 15 | 17 | 19 | 21 | 23 | 25 | 27 | 29 | 31 | 33 | 35 | 37 | 39 | 41 | 43 | 45 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 |
| t | 47 | 49 | 51 | 53 | 55 | 57 | 59 | 61 | 63 | 1 |

№4.

n, m = map(int, input().split())

A = []

for i in range(n):

A.append(i + 1)

B = [True]\*n

count = 0

i = 0

j = n

print("\nThe last two:")

while j > 0:

if B[i]: count += 1

if count == m:

if j == 1 or j == 2:

print(A[i], end=' ')

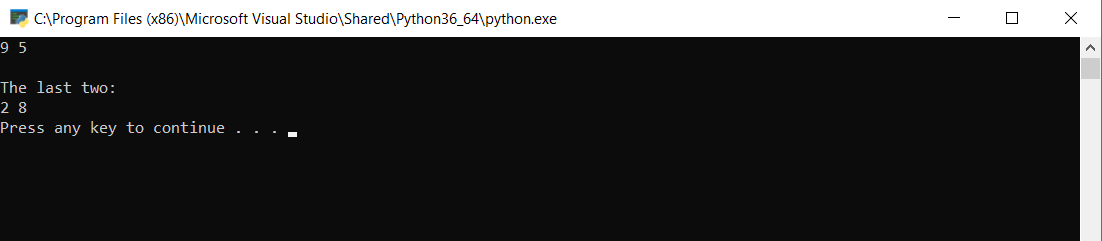
B[i] = False

count = 0

j -= 1

i = (i + 1) % n

print()



№5.

n, t = map(int, input().split())

A = []

for i in range(n):

A.append(i + 1)

B = [True]\*n

count = 0

i = 0

j = n

print("\nK:")

for k in range(1, n + 1):

while j > 0:

if B[i]: count += 1

if count == k:

if j == 1 and A[i] == t:

print(k, end=' ')

B[i] = False

count = 0

j -= 1

i = (i + 1) % n

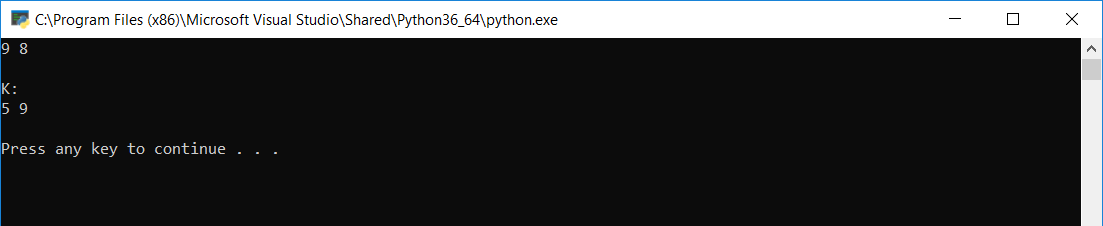
i = 0

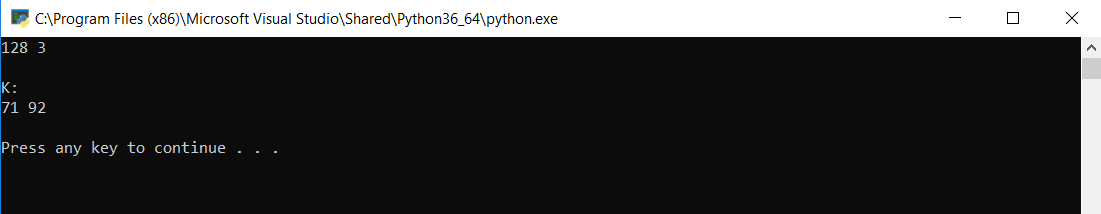
j = n

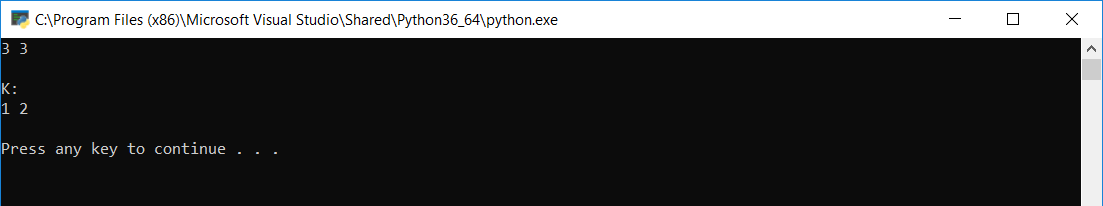
count = 0

B = [True]\*n

print('\n')







№6.

#include <iostream>

#include <ctime>

#define N 3

#define RANGE rand() % 5

struct list {

int data;

struct list \*nextPtr;

};

void listCreate(list\* &startPtr);

void printList(list\* currentPtr);

bool notDecreasing(list\* currentPtr);

int main()

{

list\* startPtr = NULL;

listCreate(startPtr);

printList(startPtr);

if (notDecreasing(startPtr))

std::cout << "list not decreases (TRUE)" << std::endl;

else

std::cout << "list decreases (FALSE)" << std::endl;

system("PAUSE");

return 0;

}

// создание списка

void listCreate(list\* &startPtr) {

srand(time(NULL));

int t = 0;

for (int i = 0; i < N; ++i) {

list\* newPtr = new list;

list\* previousPtr;

list\* currentPtr;

newPtr->data = RANGE + t;

//t += 5;

newPtr->nextPtr = NULL;

previousPtr = NULL;

currentPtr = startPtr;

while (currentPtr != NULL) {

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

if (previousPtr == NULL) {

newPtr->nextPtr = startPtr;

startPtr = newPtr;

}

else {

previousPtr->nextPtr = newPtr;

newPtr->nextPtr = currentPtr;

}

}

}

// вывод списка

void printList(list\* currentPtr) {

if (currentPtr == NULL)

puts("List is empty.\n");

else {

puts("The list is:");

while (currentPtr != NULL) {

std::cout << currentPtr->data << " --> ";

currentPtr = currentPtr->nextPtr;

}

puts("NULL\n");

}

}

// проверка на неубывание

bool notDecreasing(list\* currentPtr) {

list\* previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

while (currentPtr != NULL) {

if (previousPtr->data > currentPtr->data) {

return false;

}

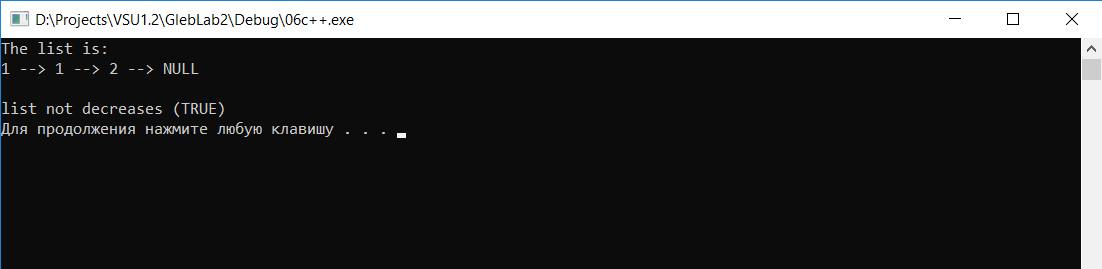
previousPtr = currentPtr;

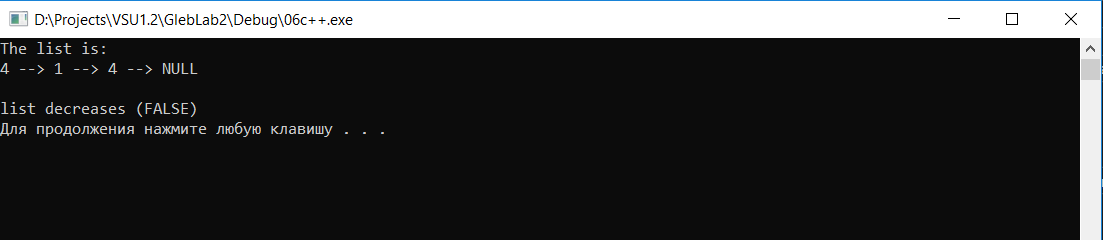
currentPtr = currentPtr->nextPtr;

}

return true;

}





№7.

#include <iostream>

#include <ctime>

#define N 5

#define RANGE rand() % 3

struct list {

int data;

struct list \*nextPtr;

};

void listCreate(list\* &startPtr);

void printList(list\* currentPtr);

list \*locate(list\* t, int x);

int main()

{

int x;

list\* startPtr = NULL;

listCreate(startPtr);

printList(startPtr);

std::cout << "Enter the X: ";

std::cin >> x;

std::cout << std::endl << "Address of first element = " << x << ":\t\t" << locate(startPtr, x) << std::endl;

std::cout << "Address of first element of list:\t" << startPtr << std::endl;

system("PAUSE");

return 0;

}

// создание списка

void listCreate(list\* &startPtr) {

srand(time(NULL));

int t = 0;

for (int i = 0; i < N; ++i) {

list\* newPtr = new list;

list\* previousPtr;

list\* currentPtr;

newPtr->data = RANGE + t;

//t += 5;

newPtr->nextPtr = NULL;

previousPtr = NULL;

currentPtr = startPtr;

while (currentPtr != NULL) {

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

if (previousPtr == NULL) {

newPtr->nextPtr = startPtr;

startPtr = newPtr;

}

else {

previousPtr->nextPtr = newPtr;

newPtr->nextPtr = currentPtr;

}

}

}

// вывод списка

void printList(list\* currentPtr) {

if (currentPtr == NULL)

puts("List is empty.\n");

else {

puts("The list is:");

while (currentPtr != NULL) {

std::cout << currentPtr->data << " --> ";

currentPtr = currentPtr->nextPtr;

}

puts("NULL\n");

}

}

// указатель на первый X

list \*locate(list\* t, int x) {

while (t != NULL && t->data != x)

t = t->nextPtr;

if (t == NULL)

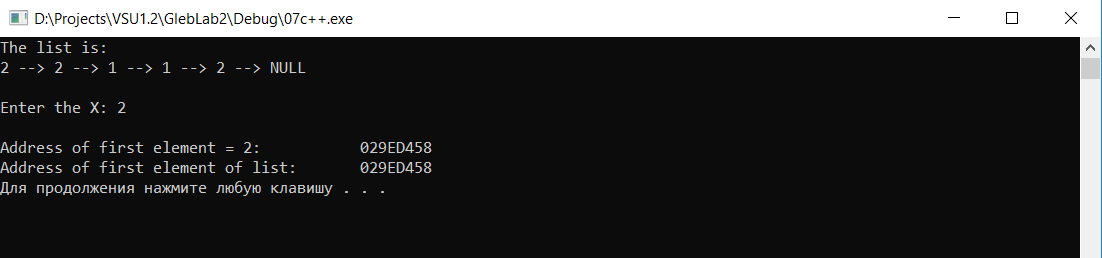
return NULL;

else

return t;

}

Для тестирования за X взят первый элемент списка, чтобы сравнить равен ли его адрес адресу списка.



№8.

Функция Retrieve возвращает позицию элемента в списке.

№9.

Функция Succ возвращает указатель на следующий элемент после p.  
В программе на слайде переменная q не изменяется, а должна в цикле while:  
q := t^.data;

№10.

Функция Pred возвращается указатель на предыдущий элемент перед p.  
Опять, в примере на слайде переменная q не изменяется, а должна в цикле while:  
q := t^.data;

№11.

#include <iostream>

#include <ctime>

#define N 5

#define RANGE rand() % 10

struct list {

int data;

struct list \*nextPtr;

};

void listCreate(list\* &startPtr);

void printList(list\* currentPtr);

double average(list\* currentPtr);

bool recursiveCheck(list\* currentPtr, int elem);

void firstLastSwap(list\* currentPtr);

int main()

{

list\* startPtr = NULL;

int elem;

listCreate(startPtr);

printList(startPtr);

std::cout << "srArif: " << average(startPtr) << std::endl << std::endl;

std::cout << "Enter the element: ";

std::cin >> elem;

if (recursiveCheck(startPtr, elem))

std::cout << elem << " is in the list" << std::endl << std::endl;

else

std::cout << elem << " is NOT in the list" << std::endl << std::endl;

firstLastSwap(startPtr);

std::cout << "AFTER SWAP:" << std::endl;

printList(startPtr);

system("PAUSE");

return 0;

}

// создание списка

void listCreate(list\* &startPtr) {

srand(time(NULL));

int t = 0;

for (int i = 0; i < N; ++i) {

list\* newPtr = new list;

list\* previousPtr;

list\* currentPtr;

newPtr->data = RANGE + t;

//t += 5;

newPtr->nextPtr = NULL;

previousPtr = NULL;

currentPtr = startPtr;

while (currentPtr != NULL) {

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

if (previousPtr == NULL) {

newPtr->nextPtr = startPtr;

startPtr = newPtr;

}

else {

previousPtr->nextPtr = newPtr;

newPtr->nextPtr = currentPtr;

}

}

}

// вывод списка

void printList(list\* currentPtr) {

if (currentPtr == NULL)

puts("List is empty.\n");

else {

puts("The list is:");

while (currentPtr != NULL) {

std::cout << currentPtr->data << " --> ";

currentPtr = currentPtr->nextPtr;

}

puts("NULL\n");

}

}

// среднее арифметическое

double average(list\* currentPtr) {

double sum = 0;

int count = 0;

while (currentPtr != NULL) {

sum += currentPtr->data;

++count;

currentPtr = currentPtr->nextPtr;

}

std::cout << "sum: " << sum << ", count: " << count << std::endl;

return sum / count;

}

// наличие элемента (рекурсивно)

bool recursiveCheck(list\* currentPtr, int elem) {

if (currentPtr->data == elem)

return true;

else if (currentPtr->nextPtr == NULL)

return false;

else

recursiveCheck(currentPtr->nextPtr, elem);

}

// перестановка первого и последнего

void firstLastSwap(list\* currentPtr) {

list\* firstPtr = currentPtr;

int temp = currentPtr->data;

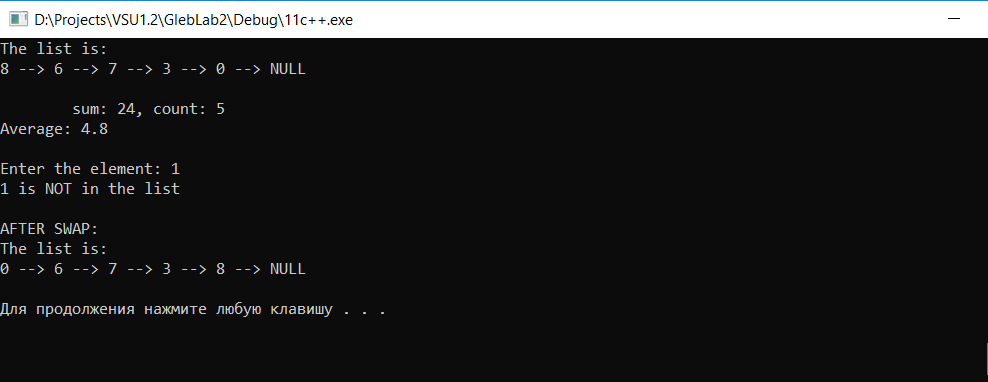
while (currentPtr->nextPtr != NULL)

currentPtr = currentPtr->nextPtr;

firstPtr->data = currentPtr->data;

currentPtr->data = temp;

}



№12.

#include <iostream>

#include <ctime>

#define N 8

#define RANGE rand() % 3

struct list {

int data;

struct list \*nextPtr;

};

void listCreate(list\* &startPtr);

void printList(list\* currentPtr);

void deleteSecondElem(list\* currentPtr);

void deleteX(list\* &currentPtr, int x);

int main()

{

list\* startPtr = NULL;

int x;

listCreate(startPtr);

printList(startPtr);

deleteSecondElem(startPtr);

std::cout << "AFTER DELETE SECOND ELEMENT: " << std::endl;

printList(startPtr);

std::cout << "Enter the X: ";

std::cin >> x;

deleteX(startPtr, x);

std::cout << "AFTER DELETE X: " << std::endl;

printList(startPtr);

system("PAUSE");

return 0;

}

// создание списка

void listCreate(list\* &startPtr) {

srand(time(NULL));

int t = 0;

for (int i = 0; i < N; ++i) {

list\* newPtr = new list;

list\* previousPtr;

list\* currentPtr;

newPtr->data = RANGE + t;

//t += 5;

newPtr->nextPtr = NULL;

previousPtr = NULL;

currentPtr = startPtr;

while (currentPtr != NULL) {

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

if (previousPtr == NULL) {

newPtr->nextPtr = startPtr;

startPtr = newPtr;

}

else {

previousPtr->nextPtr = newPtr;

newPtr->nextPtr = currentPtr;

}

}

}

// вывод списка

void printList(list\* currentPtr) {

if (currentPtr == NULL)

puts("List is empty.\n");

else {

puts("The list is:");

while (currentPtr != NULL) {

std::cout << currentPtr->data << " --> ";

currentPtr = currentPtr->nextPtr;

}

puts("NULL\n");

}

}

// удаление второго элемента

void deleteSecondElem(list\* currentPtr) {

list\* previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

list\* tPtr = currentPtr->nextPtr;

delete currentPtr;

previousPtr->nextPtr = tPtr;

}

// удаление элементов равных X

void deleteX(list\* &startPtr, int value) {

list\* cyclePtr = startPtr;

while (cyclePtr != NULL) {

list\* previousPtr;

list\* currentPtr;

list\* tempPtr;

if (value == startPtr->data) {

tempPtr = startPtr;

startPtr = startPtr->nextPtr;

delete tempPtr;

}

else {

previousPtr = cyclePtr;

currentPtr = cyclePtr->nextPtr;

while (currentPtr != NULL && currentPtr->data != value) {

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

if (currentPtr != NULL) {

tempPtr = currentPtr;

previousPtr->nextPtr = currentPtr->nextPtr;

delete tempPtr;

}

}

cyclePtr = cyclePtr->nextPtr;

}

}

