**УПРАЖНЕНИЕ 6**

**Рекурсивни функции**: Съставяне, активиране и изпълнение на рекурсивни функции. Примери за реализация. Рекурсия и итерация. **Exception Handling.**

**1. Рекурсията** е механизъм, при който една функция може да извиква сама себе си. Функциите, които използват рекурсия се наричат **рекурсивни**. Рекурсивното извикване може да бъде и индиректно – една функция вика друга, а другата функция вика първата. В този случай говорим за **косвена рекурсия. Пряка рекурсия** имаме в случая, когато тялото на една функция съдържа обръщение към същата функция. Този механизъм е възможен поради наличието на т.н. автоматични променливи (параметрите на функциите и променливите, дефинирани в тялото на функцията, които се създават при всяко нейно извикване и престават да съществуват след приключване на изпълнението й), така че при всяко извикване на функцията тя получава нов комплект автоматични променливи, който е различен от съответните комплекти за другите й извиквания.

Рекурсивното извикване може да доведе до безкраен цикъл. За да не се случи това, в тялото на функцията трябва да се проверява за тривиален случай, при който функцията връща директно стойност, а не прави само рекурсивни обръщения. При използване на рекурсивни функции, винаги трябва да се предвиди условие за изход от рекурсията, т.е. функцията да се извиква (или да не се извиква) при определено условие (нарича се дъно на рекурсията). Ако не бъде предвидено такова условие рекурсивното обръщение към функцията се превръща в безкраен цикъл, което води до препълване на стека.

Доказано е, че всяка рекурсивна функция има итеративен вариант, т.е. всеки рекурсивен алгоритъм може да се реализира без използване на рекурсия.

Рекурсивните функции са особено подходящи при реализирането на алгоритми, които са рекурсивни по определение. Типичен пример за това са алгоритмите върху дървета. Въпреки това, трябва да се има предвид, че използването на рекурсия изисква повече памет и в много случаи функциите са по-бавни от аналогични функции, реализирани без рекурсия. Предимство на рекурсивните функции е по-компактният код, който се получава. Препоръчително е за всеки отделен случай внимателно да се анализират предимствата и недостатъците от използването на рекурсия.

**Пример 1.** Да са напише рекурсивна функция, която намира факториела на въведено естествено число n. Факториел : 4! = 1.2.3.4=24

Тук тривиалният случай е (n==0)||(n==1). Тогава функцията трябва да върне стойност 1. В противен случай (ако n>1) изчисляването на факториела ще изисква рекурсивно обръщение.

#include <iostream>

using namespace std;

int fact(int n)

{

if (n == 0) return 1;

else return n \* fact(n - 1);

}

int main()

{

cout << "n= ";

int n;

cin >> n;

if (!cin || n < 0)

{

cout << "Error! \n";

return 1;

}

cout << n << "!= " << fact(n) << '\n';

system("PAUSE");

return 0;

}

В основата на реализацията на механизма на рекурсия стоят автоматичните променливи, които се създават при всяко обръщение към функцията (такива са и формалните параметри). Механизъм на действие на рекурсията:

Да разгледаме как работи тази функция за n=5. fact(5) = 5 \* fact (4)

// спира изпълнението на функцията fact за стойност n=5

// извиква се за изпълнение функция fact за стойност n=4 fact (4) = 4 \* fact(3)

// спира изпълнението за n=4

// ново извикване на fact за n=3 fact(3) = 3 \* fact(2)

//спира изпълнението за n=3

// извиква се fact за n=2 fact(2) = 2 \* fact(1)

// спира изпълнението за n=2

//извиква се fact за n=1 fact(1)=1\*fact(0)

//спира изпълнението за n=1

//пресмята се fact (0) 1 //базов случай - **дъно**

Този процес до тук се нарича **разгъване на рекурсията** (слизане към дъното). **До тука умноженията се отлагат – не се изпълняват, тъй като не знаем текущата стойност на функцията fact.** Сега започва обратния процес – **свиване на рекурсията** (нагоре). Сега се извършват изчисленията.

fact(2) = 2 \* 1=2

fact(3) = 3 \* fact(2) = 3 \* 2 = 6

fact (4) = 4 \* fact(3) = 4 \* 6 = 24

fact(5) = 5 \* fact (4) = 5 \* 24 = 120 При разгъване на рекурсията извикваща функция, текущите стойности на параметрите й 5, 4, 3 и т.н. и дефинираните в нея променливи се поставят в стек, а при свиване на рекурсията се вземат от там. Всяко извикано копие на функцията има собствена памет, затова при много рекурсивни извиквания е възможно да настъпи грешка от препълване на стека и програмата прекъсва.

За сравнение как би изглеждала задача за факториел в итеративен вариант:

#include <iostream>

using namespace std;

int main()

{

int n;

long int fak = 1;

cout << "Vyvedete chislo: ";

cin >> n;

for (int i = 1; i <= n; i++)

fak \*= i;

cout << fak << endl;

return 0;

}

Пример 2. Друг класически пример за рекурсия е пресмятането числата на

Фибоначи. Това са числата от редицата 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Всеки следващ член на тази редица е сума на предишните два члена,

т.е.

*F0 = 0*

*F1 = 1*

*Fn = Fn-1 + Fn-2* , *n > 1*

#include <iostream>

#include <windows.h>

using namespace std;

int fib(int n)

{

if (n == 0) return 0;

else if (n == 1) return 1;

else return fib(n - 1) + fib(n - 2);

}

int main()

{

SetConsoleOutputCP(1251);

cout << "Въведи число ";

double a;

cin >> a;

if (!cin || a < 0)

{

cout << "Error! \n";

return 1;

}

cout << fib(a) << endl;

system("PAUSE");

return 0;

}

**2. Exception Handling.**

Изключение е проблем, който възниква по време на изпълнението на програма. Изключението на C ++ е отговор на извънредно обстоятелство, възникващо по време на изпълнение на програма, като опит за разделяне на нула. Изключенията осигуряват начин за прехвърляне на управление от една част на програма в друга. Обработката на изключенията на C ++ е изградена върху три ключови думи: **try, catch, and throw.**

Програма хвърля изключение, когато се появи проблем. Това се прави с помощта на ключова дума throw.

catch - Програма улавя изключение с манипулатор на изключения на мястото в програма, където искате да се справите с проблема.

try - блокът try описва блок от код, за който ще бъдат активирани конкретни изключения.

Около кода се поставя блок **try / catch**, който може да генерира изключение. Кодът в блока try / catch се нарича защитен код.

Throwing Exceptions

**double division(int a, int b) {**

**if( b == 0 )**

**{**

**throw "Division by zero condition!";**

**}**

**return (a/b);**

**}**

Catching Exceptions

**try {**

**// protected code**

**} catch( ExceptionName e )**

**{**

**// code to handle ExceptionName exception**

**}**

**Пример:**

#include <iostream>

using namespace std;

double division(int a, int b) {

if( b == 0 ) {

throw "Division by zero condition!";

}

return (a/b);

}

int main () {

int x = 50;

int y = 0;

double z = 0;

try {

z = division(x, y);

cout << z << endl;

} catch (const char\* msg) {

cerr << msg << endl;

}

return 0;

}

**ЗАДАЧИ**

**Задача 1.** Напишете рекурсивна функция, която проверява дали в записа на дадено число се съдържа дадена цифра.

#include <iostream>

using namespace std;

void DigitK(int n, int k)

{

if ((n % 10) == k) cout << "Yes";

else

if (n == 0) cout << "No";

else DigitK(n / 10, k);

}

int main(void)

{

int n, k;

cout << "n = "; cin >> n;

do

{

cout << "k = ";

cin >> k;

} while ((k < 0) || (k > 9));

DigitK(n, k);

return 0;

}

**Задача 2.** Какъв ще бъде резултатът от рекурсивната функция, ако въведем x=10?

#include <iostream>

using namespace std;

void Rec(int x)

{

x = x / 2;

cout << x << " ";

if (x > 0) Rec(x);

cout << x << " ";

}

int main()

{

int x;

cout << "x= "; cin >> x;

Rec(x);

return 0;

}

**Отг. 5 2 1 0 0 1 2 5**

#include <iostream>

using namespace std;

void Rec(int **&**x)

{

x = x / 2;

cout << x << " ";

if (x > 0) Rec(x);

cout << x << " ";

}

int main()

{

int x;

cout << "x= "; cin >> x;

Rec(x);

return 0;

}

**Отг. 5 2 1 0 0 0 0 0**

**Задача 3**. Да се напише рекурсивна програма, която намира най-големия общ делител на две естествени числа.

#include <iostream>

using namespace std;

int gcd(int, int);

int main()

{

int a, b;

cout << "a= ";

cin >> a;

cout<< "b = ";

cin >> b;

if (!cin || a < 1 || b < 1)

{

cout << "Error! \n";

return 1;

}

cout << "gcd{" << a << ", " << b << "}= " << gcd(a, b) << "\n";

return 0;

}

int gcd(int a, int b)

{

if (a == b) return a; else

if (a > b) return gcd(a - b, b);

else return gcd(a, b - a);

}

**Задача 4.** Програма за взаимно рекурсивни функции.  
Двете функции rec1() и rec2() се извикват взаимно с намаляваща стъпка 2.   
#include <iostream>

using namespace std;

void rec2(int b);

void rec1(int a)

{

if (a > 0) rec2(a - 2);

cout << "a=" << a << " ";

}

void rec2(int b)

{

if (b > 0) rec1(b - 2);

cout << "b=" << b << " ";

}

int main()

{

rec1(20);

return 0;

}

**Задача 5.** Напишете рекурсивна функция, която:

a/ извежда цифрите на числото в обратен ред.

б/ извежда цифрите на числото отляво надясно.

в/ създава число със същите цифри в обратен ред.

**// а) извежда цифрите на числото в обратен ред**

#include <iostream>

using namespace std;

void Digits(int n)

{

if (n > 0)

{

cout << n % 10;

Digits(n / 10);

}

}

int main()

{

int n;

cout << "n = "; cin >> n;

Digits(n);

return 0;

}

**// б) извежда цифрите на числото отляво надясно.**

#include <iostream>

using namespace std;

void Digits(int n)

{

if (n > 0)

{

Digits(n / 10);

cout << n % 10;

}

}

int main()

{

int n;

cout << "n = "; cin >> n;

Digits(n);

return 0;

}

//**в)** **създава число със същите цифри в обратен ред.**

#include <iostream>

using namespace std;

int m = 0;

void Digits(int n)

{

if (n > 0)

{m = m \* 10 + (n % 10);

Digits(n / 10);}

}

int main()

{

int n;

cout << "n = "; cin >> n;

Digits(n);

cout << m << endl;

return 0;

}

**Задача 6.** Да се напише програма, която намира двоичното представяне на въведено естествено число.

#include <iostream>

#include<windows.h>

using namespace std;

void bin(int n)

{

if (n > 0)

{

bin(n / 2);

cout << n % 2;

}

}

int main()

{

SetConsoleOutputCP(1251);

cout << "a->";

int a;

cin >> a;

if (!cin || a < 0)

{

cout << "Error! \n";

return 1;

}

bin(a);

cout << endl;

system("PAUSE");

return 0;

}

**Задача 7.** Като се използва рекурсивната дефиниция на функцията за степенуване да се напише програма, която по дадени **a** реално и **k** – цяло число, намира стойността на ak.

#include <iostream>

using namespace std;

double pow(double, int);

int main()

{

cout << "a= ";

double a;

cin >> a;

if (!cin)

{

cout << "Error \n";

return 1;

}

cout << "k= ";

int k;

cin >> k;

if (!cin)

{

cout << "Error \n";

return 1;

}

cout << "pow{" << a << ", " << k << "}= " << pow(a, k) << "\n";

return 0;

}

double pow(double x, int n)

{

if (n == 0) return 1;

else

if (n > 0) return x \* pow(x, n - 1);

else return 1.0 / pow(x, -n);}

**Задача 8.** Да се напише програма за отпечатване на екрана на:

а) числата от 1 до N б) числата от N до 1

в) числата от N до 1 и после обратно до N г) буквите от Z до A и после обратно от А до Z

#include <iostream>

using namespace std;

//от 1 до N

void edno\_do\_N(int n)

{

if (n == 0) return;

cout << n;

edno\_do\_N(n - 1);

}

// от N до 1

void N\_do\_edno(int n)

{

if (n == 0) return;

N\_do\_edno(n - 1);

cout << n;

}

// от N до 1 и после обратно до N

void N\_edno\_N(int n)

{

if (n == 1) {

cout << 1;

return;

}

cout << n;

N\_edno\_N(n - 1);

cout << n;

}

//да се напише програма за отпечатване на буквите от Z до A и после обратно от А до Z

void Z\_A\_Z(char z, char a)

{

if (z == a) {

cout << a;

return;

}

cout << z;

Z\_A\_Z(z - 1, a);

cout << z;

}

int main(int argc, char\*\* argv)

{ int n;

cin >> n;

edno\_do\_N(n); cout << endl;

N\_do\_edno(n); cout << endl;

N\_edno\_N(n); cout << endl;

Z\_A\_Z('z', 'a');

return 0;

}

**УПРАЖНЕНИЕ 7**

**Алгоритми за сортиране на едномерен масив**:

Метод на мехурчето, метод на пряката селекция и метод на простото вмъкване.

**Алгоритми за търсене в подреден и неподреден масив**:

Двоично и последователно търсене.

1. **Алгоритми за сортиране на едномерен масив:**

**А) Метод на мехурчето.** (Bubblesort) е един от популярните и най-лесни алгоритми за сортиране. Започвайки последователно от началният елемент до крайният сравняваме всеки елемент със следващия като ги разменяме ако не са подредени. Сравнява последователно всички двойки съседни елементи

a[i-1] и а[i], и ако a[i-1] > a[i] местата им биват разменени.

По този начин на всяка стъпка изкарваме най-големият **в края** на правилното за него място. Повтаряйки този процедура толкова пъти колкото е големината на масива постигаме правилно подреждане на целият масив. Ако масивът е вече сортиран, методът на мехурчето ще премине през масива веднъж и ще установи, че не трябва да разменя никакви елементи.

3, 6, 5, 7, 1, 4, 2

Първо преглеждане по двойки:

3, 6, 5, 7, 1, 4, 2 -> 3, 6 остават така

**6**, 5 се разменят

6, 7 остава така

**7**, 1 се разменят

**7**, 4 се разменят

**7**, 2 се разменят

Масивът след първото преглеждане: 3, 5, 6, 1, 4, 2, **7** ->най големият елемент **„изплува“** най-отгоре.

Второ преглеждане по двойки..............................................

Трето преглеждане по двойки ..............................................

Накрая 1, 2, 3, 4, 5, 6, 7

**ВИДЕО:**  
<https://www.youtube.com/watch?v=lyZQPjUT5B4>

**Сортиране на масив по "Метод на мехурчето", реализиран чрез функции.**#include<iostream>

using namespace std;

int i, j, buf;

void arr\_in(int arr[], int n)

{

for (int i = 0; i < n; i++)

{

cout << "arr[" << i << "]=>";

cin >> arr[i];

}

}

void arr\_sort(int arr[], int n)

{

for (i = 1; i < n; i++)

for (j = 1; j < n; j++)

if (arr[j - 1] > arr[j])

{

buf = arr[j - 1];

arr[j - 1] = arr[j];

arr[j] = buf;

}

}

void arr\_out(int arr[], int n)

{

cout << endl;

for (int i = 0; i < n; i++)

{

cout << "arr[" << i << "]=";

cout << arr[i] << endl;

}

}

int main()

{

const int size = 6;

int a[size];

arr\_in(a, size);

arr\_sort(a, size);

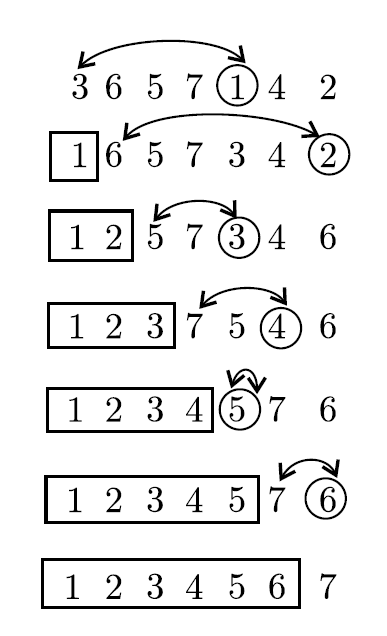
arr\_out(a, size);

return 0;

}

**Б) Сортиране чрез пряка селекция (Selection sort)*.*** Сортиране с пряка селекция Намаля броя на разменянията в сравнение с метода на мехурчето! Основна идея: Търсим най-малкият елемент в масива и директно го поставяме на първо място и т.н. Докато всички се подредят.

(Или търсим най-големия елемент и го поставяме на последно място)



#include<iostream>

using namespace std;

int i, j, buf;

void arr\_in(int arr[], int n)

{

for (int i = 0; i < n; i++)

{

cout << "arr[" << i << "]=>";

cin >> arr[i];

}

}

void arr\_sort(int arr[], int n)

{

int buf;

for (i = 0; i < n-1; i++) {

int iMin = i;

for (j = i+1 ; j < n; j++) {

if (arr[j] < arr[iMin])

iMin = j; }

if (iMin != i) {

buf = arr[iMin];

arr[iMin] = arr[i];

arr[i] = buf;

}

}

}

void arr\_out(int arr[], int n)

{

cout << endl;

for (int i = 0; i < n; i++)

{

cout << "arr[" << i << "]=";

cout << arr[i] << endl;

}

}

int main()

{

const int size = 6;

int a[size];

arr\_in(a, size);

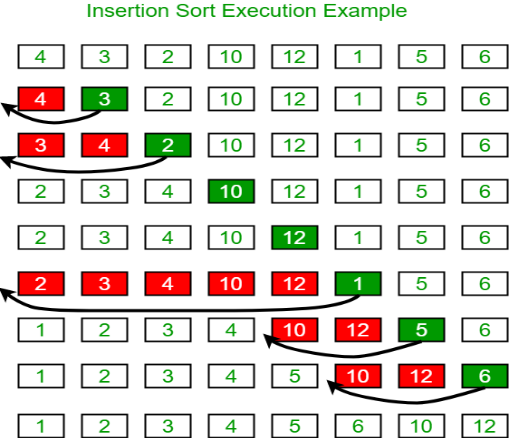
arr\_sort(a, size);

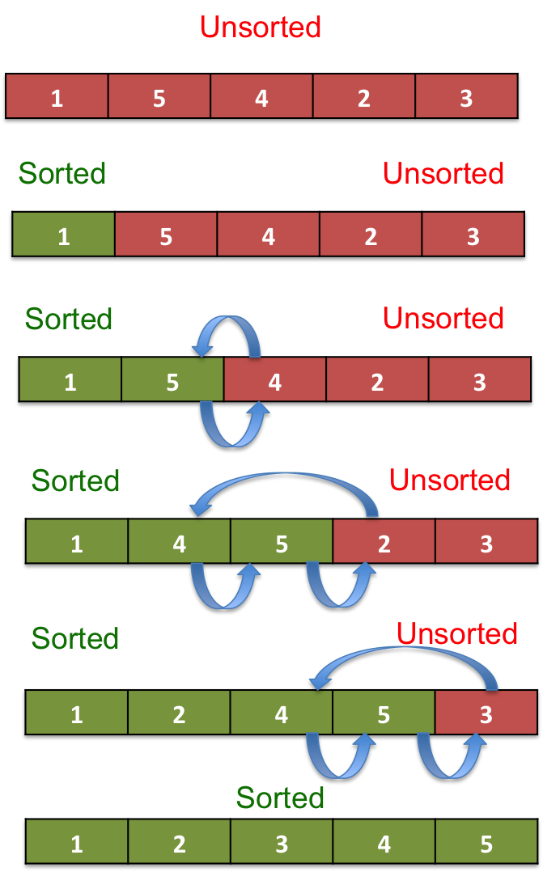
arr\_out(a, size);

return 0;

}

В) ***Сортиране чрез вмъкване*** (Insertion sort) е прост сортиращ алгоритъм. Чрез сравняващо сортиране сортираният масив се допълва с по един елемент всеки път. Масивът с елементи, които ще бъдат сортирани се разделя на две части: частта със сортираните елементи и частта с несортираните. При всяка стъпка се взема първия елемент от несортирания масив и се вмъква на правилната позиция в сортираната част от масива. Сортирането продължава докато елементите от несортираната част на масива се изчерпят. Нарича се още метод на картоиграча.



#include <iostream>

using namespace std;

void insertSort(int arr[], int size)

{

int i, j, key;

for (i = 1; i < size; i++)

{

key = arr[i];

j = i;

while ((j > 0) && (arr[j - 1] > key))

{

arr[j] = arr[j - 1];

j = j - 1;

}

arr[j] = key;

}

}

void main()

{

int a[] = {1,5,4,3,2};

insertSort(a, 5);

for (int i = 0; i < 5; i++)

cout << a[i] << endl;

}

1. **Търсене**

**А) Линейно търсене** е алгоритъм за намиране на елемент, както в сортиран, така и в несортиран масив. Идеята е много проста последователно се обхожда (може и в двете посоки) масивът и всеки елемент се сравнява с търсеният. Обхождането продължава до достигане на края на масива или до съвпадение.

#include <iostream>

using namespace std;

int lsearch(int a[], int size, int value)

{

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

if (a[i] == value)

return i;

}

return -1;

}

int main() {

const int array\_size = 8;

int list[array\_size] = { 1, 2, 3, 5, 7, 10, 14, 17 };

int search\_value;

cout << "Enter search value: ";

cin >> search\_value;

int index = lsearch(list, array\_size, search\_value);

if (index >= 0)

cout << "Search value found at index: " << index << endl;

else

cout << "Search value not found" << endl;

return 0;

}

**Б) Двойчно търсене** е алгоритъм, за намиране на елемент в **сортиран** масив.

Алгоритъма е сравнително прост и може да бъде направен както рекурсивно така и итеративно:

1) Взима средния елемент;

2) Ако средният елемент е търсената стойност, алгоритъма завършва;

3) В противен случай има 2 варианта:

- Търсената стойност е по-малка от средният елемент. В този случай стъпка 1 се повтаря с частта от масива преди средният елемент.

- Търсената стойност е по-голяма от средният елемент. В този случай стъпка 1 се повтаря с частта от масива след средният елемент.

Алгоритъма спира, когато:

а) търсеният елемент е намерен

б) когато подмасива (от ляво или дясно на средният елемент) няма повече елементи, в този случай може да заключим, че търсеният елемент го няма в масива.

#include <iostream>

using namespace std;

int bsearch(int a[], int size, int value)

{

int first, middle, last; // indexes to the array

first = 0;

last = size - 1;

while (true) {

middle = (first + last) / 2;

if (a[middle] == value)

return middle;

else if (first >= last) {

return -1;

}

else if (value < a[middle])

last = middle - 1;

else

first = middle + 1;

}

}

int main() {

const int array\_size = 8;

int list[array\_size] = { 1, 2, 3, 5, 7, 10, 14, 17 };

int search\_value;

cout << "Enter search value: ";

cin >> search\_value;

int index = bsearch(list, array\_size, search\_value);

if (index >= 0)

cout << "Search value found at index: " << index << endl;

else

cout << "Search value not found" << endl;

return 0;

}

**ЗАДАЧИ**

**Задача 1**. Да се сортира едномерен масив от цели числа във възходящ ред като се използва **методът на мехурчето**. За въвеждане и отпечатване на елементите на масива и за сортирането да се използват функции.

#include <iostream>

#include<windows.h>

using namespace std;

void bubbleSort(int numbers[], int array\_size)

{

int temp;

for (int i = 1; i < array\_size; i++)

{

for (int j = 1; j < array\_size; j++)

{

if (numbers[j - 1] > numbers[j])

{

temp = numbers[j - 1];

numbers[j - 1] = numbers[j];

numbers[j] = temp;

}

}

}

}

void read(int numbers[], int array\_size)

{

for (int i = 0; i < array\_size; i++)

{

cout << "Въведете елемент "<< i << ":";

cin >> numbers[i];

}

}

void write(int numbers[], int array\_size)

{

for (int i = 0; i < array\_size; i++)

{

cout << numbers[i] << " ";

}

}

int main()

{

SetConsoleOutputCP(1251);

int n;

do

{

cout << "Въведете брой на елементите на масива: ";

cin >> n;

} while (n < 1);

int a[20];

read(a, n);

bubbleSort(a, n);

write(a, n);

system("pause");

return 0;

}

**Задача 2.** Да се сортира масив от 10 случайно генерирани цели числа в интервала от 0 до 100 в низходящ ред като се използва **пряка селекция**. Да се напишат функции за генериране на числата, извеждане на елементите на масива и за сортиране.

#include <iostream>

#include<ctime>

#include<windows.h>

using namespace std;

void SelectSort(int A[], int n)

{

int max;

int b;

for (int a = 0; a < n - 1; a++)

{

b = a;

max = A[b];

for (int j = a + 1; j < n; j++)

{

if (A[j] > max)

{

b = j;

max = A[b];

}

}

A[b] = A[a];

A[a] = max;

}

}

void read(int numbers[], int array\_size)

{

srand(static\_cast<unsigned>(time(NULL)));

for (int i = 0; i < array\_size; i++)

{

numbers[i] = rand() % (100);

}

}

void print(int numbers[], int array\_size)

{

for (int i = 0; i < array\_size; i++)

{

cout << numbers[i] << " ";

cout << endl;

}

}

int main()

{

SetConsoleOutputCP(1251);

int a[10];

read(a, 10);

cout << "Изход преди сортиране: " << endl;

print(a, 10);

SelectSort(a, 10);

cout << "Изход след сортиране: " << endl;

print(a, 10);

system("pause");

return 0;

}

**Задача 3.** Да се сортира масив от 10 случайно генерирани цели числа в интервала от 0 до 100 във възходящ ред като се използва сортиране чрез **вмъкване**. Да се напишат функции за генериране на числата, извеждане на елементите на масива и за сортиране.

#include <iostream>

#include<ctime>

#include<windows.h>

using namespace std;

void InsertionSort(int A[], int n)

{

int k, index;

for (int j = 1; j < n; j++)

{

index = A[j];

k = j;

while (k > 0 && (A[k - 1] > index))

{

A[k] = A[k - 1];

k = k - 1;

}

A[k] = index;

}

}

void read(int numbers[], int array\_size)

{

srand(static\_cast<unsigned>(time(NULL)));

for (int i = 0; i < array\_size; i++)

{

numbers[i] = rand() % (100);

}

}

void print(int numbers[], int array\_size)

{

for (int i = 0; i < array\_size; i++)

{

cout << numbers[i] << " ";

cout << endl;

}

}

int main()

{

SetConsoleOutputCP(1251);

int a[10];

read(a, 10);

cout << "Изход преди сортиране: " << endl;

print(a, 10);

InsertionSort(a, 10);

cout << "Изход след сортиране: " << endl;

print(a, 10);

system("pause");

return 0;}

**Задача 4.** Да се напише функция, която реализира **линейно търсене** на елемент в масив от цели числа.

#include <iostream>

using namespace std;

int linear\_search(int data[], int size, int value)

{

int i = 0;

while (i < size && data[i] != value)

{

i++;

};

if (i < size) return i;

else return -1;

}

int main() {

const int n = 5;

int numbers[n] = { 2,5,67,8,9 };

int search\_value;

cout << "Enter search value: ";

cin >> search\_value;

int index = linear\_search(numbers, n, search\_value);

if (index >= 0) cout << "Search value found at index: " << index << endl;

else cout << "Search value not found, index returned: " << index << endl;

system("pause");

return 0;

}

**Задача 5.** Да се напише рекурсивна функция, която търси въведена стойност в масив от цели числа като се използва алгоритъмът за **двоично търсене**. (рекурсивен вариант)

#include <iostream>

using namespace std;

int bsearch(int a[], int from, int to, int n)

{

if (from > to)

return -1;

else

{

int mid = (from + to) / 2;

if (a[mid] == n)

return mid;

else if (a[mid] < n)

return bsearch(a, mid + 1, to, n);

else

return bsearch(a, from, mid - 1, n);

}

}

int main()

{

int numbers[5] = { 2,3,5,8,9 };

int search\_value;

cout << "Enter search value: ";

cin >> search\_value;

int index = bsearch(numbers, 0, 4, search\_value);

if (index >= 0) cout << "Search value found at index: " << index << endl;

else cout << "Search value not found, index returned: " << index << endl;

system("pause");

return 0;

}

Допълнение: използване на указатели при метода на мехурчето.

**Зад. Да се напише програма, в която чрез отделни функции:**

** се въвеждат елементи на едномерен целочислен масив;**

** сортира се масива във възходящ ред;**

** извеждат се сортираните елементи.**

#include <iostream>

using namespace std;

//Функция за въвеждане елементи на едномерен целочислен масив

void input(int\* a, int n)

{

for (int i = 0; i < n; i++)

{

cout << "a [" << i << "] =";

cin >> \*(a + i);

}

}

//Функция за извеждане елементи на едномерен целочислен масив

void output(int\* a, int n)

{

for (int i = 0; i < n; i++)

cout << "a [" << i << "] =" << \*(a + i) << endl;

}

//Функция за сортиране по метод на мехурчето чрез използване на указатели.

void sort(int\* array, int num)

{

int i, j;

int temp; /\* Used in swopping array values \*/

for (i = 0; i < num; i++)

for (j = 0; j < num - 1; j++)

if (\*(array + j) > \*(array + j + 1))

{

temp = \*(array + j);

\*(array + j) = \*(array + j + 1);

\*(array + j + 1) = temp;

}

}

int main()

{

const int N = 50; int n;

int a[N];

do

{cout << "n = "; cin >> n;

} while (n < 1 || n>50);

input(a, n); // ПП за въвеждане

sort(a, n);

cout << "Sortiran masiv : " << endl;

output(a, n);

}

**УПРАЖНЕНИЕ 8.1**

**Низове**: Обработка на низове. Копиране, сливане, сравняване на низове. Стандартни функции за обработка на низове с примери. **Класът String.**

**Знаков (символен) низ** – крайна редица от знакове (символи), заградени в кавички. Броят на символите в редицата се нарича дължина на низа. Низ с дължина 0, се нарича празен низ.

Примери: “ xyz” - знаков низ с дължина 3

“This is a string.” – низ с дължина 17

“” – празен низ

Низ, който се съдържа в даден низ, се нарича негов **подниз**.

Ще разгледаме две от възможностите на езика C++ за работа със символни низове:

- представени като масиви от символи от тип **char** (C-style strings);

- като обекти на класа **string** (string objects).

1. **Дефиниране и инициализиране на низ от тип char.**

Символният низ е едномерен масив, който завършва с нулев символ ‘\0’.

**Дефинирането** на низ става чрез:

* низови константи
* едномерни масиви от тип char
* указатели от тип char
* масиви от символни низове

1. **Низови константи**

Низова (символна) константа се задава като последователност от символи, заградени с двойни кавички. Компилаторът записва символите в последователни клетки, като накрая прибавя символа ’\0’.

char s1[]= “ This is a string.“;

Не се указва максималния размер на масива. Компилаторът преброява символите, поставя ‘\0’ и отделя памет за масива.

сhar s1[]={‘ ‘, ‘t’,’h’,’i’,’s’,’ ‘,’i’,’s’,’ ‘,’a’,’ ‘,’s’,’t’,’r’,’i’,’n’,’g’,’.’,’\0’};

1. **Едномерен масив от тип char**

char str1[20] ;

В размерността трябва да се предвиди и един елемент за символа ‘\0’.

char str2[5] = {‘a’, ‘b’, ‘c’};

Дефинира масив и го инициализира с 3 символа, останалите са ‘\0’.

char str2[5]= {‘a’, ‘b’,’c’,’\0’,’\0’};

Данните се въвеждат по време на работа на програмата.

1. **Указател от тип char**

char \*str1=”this is a string.”;

Инициализира се променливата от тип указател, който сочи към низа, записан в паметта. Броят на символите в низовата константа определя големината на паметта.

Достъпът до отделните символи на низа е възможен чрез индекси или чрез указателната нотация:

char name[] = "Victoria I. Doncheva";

cout << name << endl;

name[8] = '\0'; //индексна операция

cout << name << endl;

\*(name+8) = ' '; //указателна нотация

cout << name << endl;

1. **Операции, релации и вградени функции:**

* **Въвеждане и извеждане**: Низовете могат да се въвеждат и извеждат с помощта на cin и cout, както и с помощта на printf и scanf.
* **Всички останали операции и релации** с низове са реализирани като стандартни функции, чиито прототипи се намират в **string. h. (string)**

1. **Въвеждане на низ.**

char name[25];

**cin** >> name;

Въведеното име не трябва да надвишава 24 символа, за да има място за

специалния символ за край. Това може да се контролира с манипулатора

setw: cin >> setw(25) >> name;

Ако бъдат въведени повече от 24 символа, всички след 24-тия ще бъдат игнорирани. Трябва да се има предвид, че знаците **интервал, табулация или** **натиснат клавиш Enter** се възприемат като край на символния низ, т.е. не можем да въведем низ, съдържащ тези знаци. В някои случаи обаче, е добре да могат да се въвеждат и такива знаци, например, ако искаме да въведем име и фамилия в един низ. За тази цел въвеждането трябва да стане с функцията **cin.getline().** Тя има три параметъра – **масив от символи**, **дължина** и **символ за край** на въвеждането. Последният параметър не е задължителен (той има подразбиращасе стойност **'\n'**). Ако той се пропусне, в низа ще се запише всичко въведено, до натискане на Enter:

cin.getline(name, 24);

Ако е включен и третият параметър, въвеждането не се прекратява с Enter, a продължава докато се въведе символът за край и чак след това Enter.

Например, ако при оператора

cin.getline(name, 24, '.');

въведем от клавиатурата

Victoria I. Doncheva Enter

в низа name ще се запише само

Victoria I

Общ вид:

**cin.getline(<var\_str>,<size>,[<char>] )**

var\_str – променлива от тип низ;

size – цял израз, максимален размер на низа (size –1)

char – произволен символ

Въвежда се от буфера на клавиатурата редица от знаци (символи) с максимален размер на низа (size – 1). Въвеждането продължава до срещане на символа зададен в char, или до въвеждане на (size – 1) символа.

**Примери:**

char s1[10];

cin.getline(s1,10);

Очаква се въвеждане на низ, Enter size =9 + ‘\0’

Aкo e въведен по-голям низ, се вземат първите 9 символа, свързват се с s1 и изпълнението завършва преди Enter.

1. **Извеждане на низ.**

**cout**<<s;

Знакът за край на низ идентифицира края му.

Трябва да се отбележи, че използването на тези функции изисква включване на заглавен файл ***string.h***: #include <string.h>

1. **Дължина на низ.**

**strlen (<str>)**

<str> е произволен низ (константа, променлива или израз със стойност символен низ

Намира дължината на низа <str>.

strlen(“abc”) = 3

**Пример. Да се намери дължината на низа name:**

#include <iostream>

#include <string.h>

#include <windows.h>

using namespace std;

int main()

{

SetConsoleOutputCP(1251);// за изход

SetConsoleCP(1251); // за вход

char name[20] = "Ivan Petrov";

cout << "Низа : " << name << " има дължина " << strlen(name);

cout << " символа. ";

return 0;

}

1. **Конкатенация на низове.**

Конкатенация на два низа е, низ който е получен, като в края на първият низ се запише вторият. Нарича се още слепване на низове.

**strcat (<var\_str>, <str>)**

<var\_str> - променлива от тип низ

<str> низ (константа, променлива , израз със стойност символен низ)

Конкатенира низа от <var\_str> с низа <str>, като **резултатът се съдържа в променливата <var\_str>.**

**Пример: Да се намери пълното име name при зададени малко име firstName и фамилия familyName:**

#include <iostream>

#include <string.h>

using namespace std;

int main() {

char firstName[20] = "Ivan", familyName[20] = "Petrov";

char name[20] = "";

strcat(name, firstName);

strcat(name, " ");

strcat(name, familyName);

cout << name << endl;

return 0;

}

1. **Сравняване на низове**

**strcmp (<str1>, <str2>)**

Функцията е целочислена като резултатът е:

-1 <str1> **<** <str2>

0 <str1> **=** <str2>

1. <str1> **>** <str2>
2. **Копиране на низове.**

**strcpy(<var\_str>, <str>)**

<var\_str> - променлива от тип низ

<str> низ (константа, променлива , израз със стойност символен низ)

**strcpy(низова\_променлива,низ)**Функцията strcpy копира низ в низова\_променлива. Ако низът е по-дълъг от низовата променлива са възможни труднооткриваеми грешки. Резултатът от копирането се връща от функцията, а също се съдържа и в низова\_променлива.  
**Пример. Следващият код извежда низа "Ivan", който е копиран от firstName в променливата name:**

#include <iostream>

#include <string.h>

using namespace std;

int main() {

char firstName[5] = "Ivan";

char name[20];

strcpy(name, firstName);

cout << name << endl;

return 0;

}

1. **Търсене на низ в друг низ.**

**strstr(<str1>,<str2>)**

Търси str2 в низа str1. Ако str2 се съдържа в str1, функцията връща подниза на str1, започващ от първото срещане на str2 до края на низа str1. Ако str2 не се съдържа връща 0-лев указател.

Примери: Програмният фрагмент

char str1[15] = "asemadaemada", str2[10]= "ema";

cout << strstr(str1, str2) << "\n";

извежда

emadaemada

а

char str1[15] = "asemadaemada", str2[10]= "ema";

cout << strstr(str2, str1) << "\n";

предизвиква съобщение за грешка по време на изпълнение.

1. **Преобразуване между тип низ и други стандартни типове** – стандартни функции, реализирани в **<stdlib.h>.**

|  |  |
| --- | --- |
| **Функция** | **Предназначение** |
| atof | Преобразува string в **float** |
| atoi | Преобразува string в **int** |
| atol | Преобразува string в **long** |
| itoa | Преобразува **int** в string |
| ltoa | Преобразува **long** в string |
| ecvt | Преобразува **double** в string |
| strtod | Преобразува string в **double** |

1. **Конкатенация на n символа от низ с друг низ.**

**strncat(<var-str>, <str>, n)**

<var\_str> е променлива от тип низ,

<str> е низ (константа, променлива или по-общо израз), а

n е цял израз с неотрицателна стойност.

Семантика

Копира първите n символа от <str> в края на низа, който е стойност на <var\_str>. Копирането завършва когато са прехвърлени n символа, или е достигнат краят на <str>. Резултатът е в променливата <var\_str>. За използване на тази функция е необходимо да се включи заглавният файл string.h.

Пример: Резултатът от изпълнението на фрагмента:

chat a[10] = “aaaaa”;

strncat(a, “qwertyqwerty”, 5);

cout << a;

е

aaaaaqwert

а на

strncat(a, “qwertyqwerty”, -5);

cout << a;

предизвиква съобщение за грешка.

1. **Копиране на *п* символа на низ в друг низ.**

**strncpy(<var\_str>,<str>,n)**

<var\_str> е променлива от тип низ, <str> е низ (константа, променлива или по-общо израз), а n е цял израз с неотрицателна стойност.

Семантика

Копира първите n символа на <str1> в <var\_str>. Ако <str> има по-малко от n символа, ‘\0’ се копира до тогава докато не се запишат n символа. Параметърът <var\_str> трябва да е от вида char[n] и съдържа резултатния низ. За използване на тази функция е необходимо да се включи заглавният файл string.h.

Примери: 1. Програмният фрагмент

char a[10];

strncpy(a, "1234567", 8);

cout << a << "\n";

извежда

1234567

Изпълнява се по следния начин: тъй като дължината на низа “1234567” е

по-малка от 8, допълва се с един знак ‘\0’ и се свързва с

променливата a.

2. Програмният фрагмент

char a[10];

strncpy(a, "123456789", 5);

cout << a << "\n";

извежда

12345<неопределено>

Изпълнява се по следния начин: тъй като дължината на низа “123456789”

е по-голяма от 5, низът “12345” се свързва с променливата a, но не

става допълване с ‘\0’, което личи по резултата.

1. **Сравняване на *n* символа на низове.**

**strncmp(<str1>,<str2>,n)**

<str1> и <str2> са низове (константи, променливи или по-общо изрази), а n е цял израз с неотрицателна стойност.

Семантика

Сравнява първите n символа на <str1> със символите от съответната позиция на <str2>. Сравнението продължава до намиране на два различни символа или до края на един от символните низове. Резултатът от функцията strncmp е цяло число с отрицателна стойност, ако <str1> е по-малък от <str2>, 0 – ако <str1> е равен на

<str2> и с положителна стойност, ако <str1> е по-голям от <str2>. За използване на strncmp е необходимо да се включи заглавният файл string.h.

Примери:

1. char a[10] = “qwer”, b[15] = “qwerty”;

if (!strncmp(a, b, 3)) cout << “yes \n”;

else cout << “no \n”;

извежда yes, тъй като strncpm(a, b) връща 0 (низовете са равни),

!strncmp(a, b) e 1 (true).

2. char a[10] = “qwer”, b[15] = “qwerty”;

if (strncmp(a, b, 5)) cout << “yes \n”;

else cout << “no \n”;

извежда yes, тъй като strncpm(a, b) връща -1 (a е по-малък от b).

3. char a[10] = “qwerty”, b[15] = “qwer”;

if (strncmp(a, b, 5)) cout << “yes \n”;

else cout << “no \n”;

извежда yes, тъй като strncpm(a, b) връща 1 (a е по-голям от b).

1. **Търсене на символ в низ.**

**strchr(<str>,<expr>)**

<str> е произволен низ, а

<expr> e израз от интегрален или изброен тип с положителна

стойност, означаваща ASCII код на символ.

Семантика

Търси първото срещане на символа, чийто ASCII код е равен на стойността на <expr>. Ако символът се среща, функцията връща подниза на <str> започващ от първото срещане на символа и продължаващ до края му. Ако символът не се среща - връща “нулев указател”. “нулев указател”. Последното означава, че в позиция на условие, функционялното обръщение ще има стойност false, но при опит за извеждане, ще предизвика грешка.

Примери: Операторът

cout << strchr(“qwerty”, ‘e’);

извежда

erty

Операторът

cout << strchr(“qwerty”, ‘p’);

извежда съобщение за грешка, а

if (strchr(“qwerty”, ‘p’)) cout << “yes \n”; else cout << “no \n”;

извежда no, тъй като ‘p’ не се среща в низа “qwerty”.

1. **Търсене на първата разлика.**

**strspn(<str1>, <str2>)**

<str1> и <str2> са произволни низове (константи, променливи или по-общо изрази).

Семантика

Проверява до коя позиция <str1> и <str2> съвпадат. Връща дължината на префикса до първия различен символ. За използване на тази функция е необходимо да се включи заглавният файл string.h.

Пример: Програмният фрагмент

char a[10]= "asdndf", b[15] = "asdsdfdhf";

cout << strspn(a, b) << "\n";

извежда 3 тъй като първият символ, по който се различават a и b е в позиция 4.

**III.Тип string в програмен език С++.**

**Низовете, като обекти на класа string**

Стандартната С++ библиотека поддържа тип ***string***. Дефинирането на символен низ може да бъде по следния начин:

**string str1 = ”Hello”;**

**string str2 = ”World”;**

**string str3;**

В случая, дефинирани са три променливи от тип низ като първите две са инициализинари с посочните низови константи. Третата променлива не е инициализирана.

Допустимо е присовояване както на стойност низова константа, така и присвояването на стойност от друга низова променлива. Например, следващият програмен ред ще копира стойността ”Hello” на променливата str3.

str3=str1;

Оператор + (събиране) може да се използва с операнди низове като резултатът е слепване на двата низа. В случая, този оператор се нарича **конкатенация**.

Пример:

str3=str1+str2;

cout<<str3;

В случая, на екрана ще се изведе съобщението ”HelloWorld”.

Низовете могат да бъдат операнди и в операторите за сравнение <, >, <=, >=, ==, !=.

Чрез тях се сравнява съдържанието на два низа. Както и в език С, допустимо е да се извлече символ от низ като се използва индексиране.

Пример:

string city = ”Gabrovo”;

char firtSymbol=city[0];

cout<<firstSymbol;

В посочения пример променливата firtSymbol получава стойност G и този символ се извежда на екрана.

**Какви са предимствата и недостатъците на използването на низовия клас спрямо char \*?** Създаването на новия тип низ се дължи на недостатъците при работа със символни низове, които char \* тип демонстрира. В сравнение с типа char \*, типът низ има следните основни предимства:

- **възможността за обработка на низове с помощта на стандартни C ++ оператори (=, +, = =, <> и т.н.). Както знаете, когато използвате типа char \*, дори най-простите операции с низове изглеждаха сложни и изискваха писане на прекомерен програмен код;**

**- осигуряване на по-добра надеждност (сигурност) на програмния код. Например, когато копирате низове, типът низ предоставя подходящите действия, които могат да възникнат, ако изходният низ е по-голям от целевия низ;**

**- предоставяне на низ като независим тип данни. Декларацията на типа низ като низ е еднаква за всички променливи в програмата, което осигурява последователност на данните.**

Основният недостатък на типа низ в сравнение с типа char \* е по-ниската скорост на обработка на данните. Това е така, защото типът низ всъщност е клас на контейнер. А работата с клас изисква допълнително внедряване на програмния код, което от своя страна отнема допълнително време.

Следните оператори могат да се използват с обекти от низовия клас

= - задание

+ - конкатенация (конкатенация на низове)

+ = - задание с конкатенация

== - равенство

! = - неравенство

< - по-малко

<= - по-малко или равно

> - още

> = - повече или равно

[] - индексиране

Пример, който демонстрира използването на горните оператори

// тип низ, операции над низове

string s1 = "s-1";

string s2 = "s-2";

string s3;

bool b;

// операция '=' (присвояване на низове)

s3 = s1; // s3 = "s-1"

// операция '+' - конкатенация на низове

s3 = s3 + s2; // s3 = "s-1s-2"

// операция '+ =' - присвояване с конкатенация

s3 = "s-3";

s3 + = "abc"; // s3 = "s-3abc"

// операция '==' - сравнение на низове

b = s2 == s1; // b = false

b = s2 == "s-2"; // b = вярно

// операция '! =' - сравнение на низове (не е равно)

s1 = "s1";

s2 = "s2";

b = s1! = s2; // b = вярно

// операции '<' и '>' - сравнение на низове

s1 = "abcd";

s2 = "de";

b = s1> s2; // b = false

b = s1 <s2; // b = вярно

// операции '<=' и '> =' - сравнение на низове (по-малко или равно, по-голямо или равно)

s1 = "abcd";

s2 = "ab";

b = s1> = s2; // b = вярно

b = s1 <= s2; // b = false

b = s2> = "ab"; // b = вярно

// операция [] - индексиране

char c;

s1 = "abcd";

c = s1 [2]; // c = 'c'

c = s1 [0]; // c = 'a'

1. **Въвеждане и извеждане на низ в класа String.**

Няма разлика и във въвеждането на стойност за низа от клавиатурата:

**cin >> s1;**

По този начин не е възможно въвеждането на низ, съдържащ интервали, за това в повечето случаи е добра идея да използваме за тази цел функцията getline():

**getline(cin, s1);**

Така въвеждането продължава до натискането на клавиш *Enter*, което означава, че низът не може да съдържа символа '\n' за нов ред. Това ограничение също може да се избегне с третия параметър на getline():

getline(cin, s1, '\*');

Сега въвеждането продължава, докато се срещне символът '\*'.

Ако при дефиниране на низ не се посочи инициализатор, низът ще бъде празен:

string s2;

Дължината на такъв низ е 0.

Извеждането на самия низ на екрана става по същия начин, както и извеждането на променливи от познатите ни типове:

**cout << s1 << endl;**

1. **Присвояване на низове. Assign().**

За да присвоите един низ на друг, можете да използвате един от двата метода:

- използвайте оператора за присвояване '=';

- използвайте функцията assign () от класа string.

Функцията **assign ()** има няколко реализации. Първата опция е да извикате функция без параметри:

**string & assign (void);**

В този случай има просто присвояване на един низ на друг.

Втората опция ви позволява да копирате даден брой знаци от низ:

**string &assign(const string & s, size\_type st, size\_type num);**

Където s - обектът, от който е взет изходният низ;

st - индекс (позиция) в низа, от който да започне копирането на num символи;

num - броят на знаците за копиране от позиция st;

size\_type е пореден тип данни.

Третият вариант на функцията assign () копира първите числови символи от низ s на повикващия: **string & assign(const char \* s, size\_type num);**

Където s - низ, който завършва със знака '\ 0';

num е броят на символите, които се копират в повикващия. Първите num символи от низ s се копират.

По-долу е даден пример с различни реализации на функцията assign ().

Пример.

// присвояване на низове, функция assign ()

string s1 = "bestprog.net";

string s2;

string s3;

char \* ps = "bestprog.net";

s3 = s1; // s3 = "bestprog.net"

s2.assign(s1); // s2 = "bestprog.net"

s2.assign(s1, 0, 4); // s2 = "best"

s2.assign(ps, 8); // s2 = "bestprog"

1. **Дължина на низа.** Функцията **length ().**

низ txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

cout << "Дължината на txt низа е:" << txt.length ();

Съвет: Може да видите някои програми на C ++, които използват функцията size (), за да получат дължината на низ. Това е само псевдоним на length (). Изцяло зависи от вас дали искате да използвате length () или size ():

Пример

низ txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

cout << "Дължината на txt низа е:" << txt.size ();

1. **Обединяване на низове. Функция Append ().**

Функцията **append ()** се използва за обединяване на низове. Можете също да използвате операцията „+“, за да добавите редове, например:

string s1;

string s2;

s1 = "abc";

s2 = "def";

s1 = s1 + s2; // s1 = "abcdef"

Функцията append () обаче е добра, ако трябва да добавите част от низ.

Функцията има следните опции за изпълнение:

**string &append(const string & s, size\_type start);**

**string &append(const char \* s, size\_type num);**

При първото изпълнение функцията получава препратка към низовия обект s, който се добавя към повикващия. Във второто изпълнение функцията получава указател към низ const char \*, който завършва със символа '\ 0'.

Пример. Демонстрация на функцията append ().

string s1 = "abcdef";

s2 = "1234567890";

append(s2, 3, 4); // s1 = "abcdef4567"

char \* ps = "1234567890";

s1 = "abcdef";

s1.append(ps, 3); // s1 = "abcdef123"

1. **Вмъкване на символи в ред. Функция Insert ().**

За да вмъкнете един ред в дадена позиция на друг ред, трябва да използвате функцията **insert (),** която има няколко опции за изпълнение.

Първата версия на функцията ви позволява да вмъкнете целия низ s в дадената начална позиция на извикващата линия (извикващ обект):

**string & insert(size\_type start, const string &s);**

Втората версия на функцията ви позволява да вмъкнете част (insStart, num параметри) от низ s в дадената начална позиция на извикващия низ:

**string & insert(size\_type start, const string &s, size\_type insStart, size\_type num);**

В горните функции:

s - низът, който трябва да се вмъкне в повикващия низ;

start - позицията в извикващата линия, от която да се вмъкне низът s;

insStart - позицията в низ s, от която да се вмъкне;

num е броят на символите в низ s, които са вмъкнати от позиция insStart.

string s1 = "abcdef";

string s2 = "1234567890";

s1.insert(3, s2); // s1 = "abc"+"1234567890"+"def"="abc1234567890def"

s2.insert(2, s1, 1, 3); // s2 = "12bcd34567890"

1. **Замяна на символи в низ. Функция замяна ().**

Функцията **replace ()** замества символите в извикващия низ. Функцията има следните опции за изпълнение:

**string &replace(size\_type start, size\_type num, const string &s);**

**string &replace(size\_type start, size\_type num, const string &s, size\_type replStart, size\_type replNum);**

При първата реализация извикващият низ се заменя със низ s. Възможно е да зададете позицията (начало) и броят на символите (число) в повикващата линия, които трябва да бъдат заменени с низа s.

Втората версия на функцията replace () се различава от първата по това, че ви позволява да замените извикващия низ само с част от низа s. В този случай се дават два допълнителни параметъра: позицията на replStart и броят на символите в низа s, които образуват подниза, който замества извикващия низ.

Пример. Демонстрация на функцията replace ().

string s1 = "abcdef";

string s2 = "1234567890";

s2.replace(2, 4, s1); // s2 = "12abcdef7890"

s2 = "1234567890";

s2.replace(3, 2, s1); // s2 = "123abcdef67890"

s2 = "1234567890";

s2.replace(5, 1, s1); // s2 = "12345abcdef7890"

// замена символов, функция replace()

string s1 = "abcdef";

string s2 = "1234567890";

s2.replace(2, 4, s1); // s2 = "12abcdef7890"

s2 = "1234567890";

s2.replace(3, 2, s1); // s2 = "123abcdef67890"

s2 = "1234567890";

s2.replace(5, 1, s1); // s2 = "12345abcdef7890"

s2 = "1234567890";

s2.replace(5, 1, s1, 2, 3); // s2 = "12345cde7890"

s2 = "1234567890";

s2.replace(4, 2, s1, 0, 4); // s2 = "1234abcd7890"

1. **Премахнете определен брой символи от низ. Функция erase().**

За да премахнете символите от извикващия низ, използвайте функцията **erase ():**

**string & erase(size\_type index=0, size\_type num = npos);**

Където индекс - индекс (позиция), започвайки от който искате да премахнете символите в извикващата линия;

num е броят на символите, които трябва да бъдат премахнати.

Пример.

string s = "01234567890";

s.erase(3, 5); // s = "012890"

s = "01234567890";

s.erase(); // s = ""

1. **Търсете символ в низ. find () и rfind ().**

В низовия клас търсенето на низ в подниз може да се извърши по два начина, които се различават в посоката на търсене:

чрез разглеждане на низа от началото до края с помощта на функцията find ();

чрез сканиране на низа от края до началото с rfind ().

Прототипът на функцията find () е:

**size\_type find(const string &s, size\_type start = 0) const;**

Където s е поднизът, който се търси в низа, който извиква дадената функция. Функцията търси първата поява на низа s. Ако поднизът s е намерен в низа, който извиква дадената функция, тогава се връща позицията на първото появяване. В противен случай се връща -1;

старт - позицията, от която се извършва търсенето.

Прототипът за функцията rfind () е:

**size\_type rfind(const string &s, size\_type start = npos) const;**

Където s е поднизът, който трябва да се търси в извикващия низ. Търсенето на подниз в низ се извършва от край до началото. Ако поднизът s е намерен в извикващия низ, тогава функцията връща позицията на първото появяване. В противен случай функцията връща -1;

npos - позиция на последния знак на извикващата линия;

старт - позицията, от която се извършва търсенето.

Пример 1. Фрагмент от код, който демонстрира резултата от функцията find ()

// тип string, функция find()

string s1 = "01234567890";

string s2 = "345";

string s3 = "abcd";

int pos;

pos = s1.find(s2); // pos = 3

pos = s1.find(s2, 1); // pos = 3

pos = s1.find("jklmn", 0); // pos = -1

pos = s1.find(s3); // pos = -1

pos = s2.find(s1); // pos = -1

**Пример 2.** Работа на функцията rfind().

// тип string, функциите find() и rfind()

string s1 = "01234567890";

string s2 = "345";

string s3 = "abcd";

string s4 = "abcd---abcd";

int pos;

pos = s1.rfind(s2); // pos = 3

pos = s1.rfind(s2, 12); // pos = 3

pos = s1.rfind(s2, 3); // pos = 3

pos = s1.rfind(s2, 2); // pos = -1

pos = s2.rfind(s1); // pos = -1

pos = s1.rfind(s3, 0); // pos = -1

// разлика между find() и rfind()

pos = s4.rfind(s3); // pos = 7

pos = s4.find(s3); // pos = 0

1. **Сравнение на части от низове. Функция Compare ().**

Тъй като типът низ е клас, можете да използвате операцията '= =', за да сравните два низа помежду си. Ако двата низа са еднакви, сравнението ще бъде вярно. В противен случай сравнението ще бъде невярно.

Но ако трябва да сравните част от един низ с друг, тогава за това е предвидена функцията **compare ().**

Прототипът на функцията compare ():

**int compare(size\_type start, size\_type num, const string &s) const;**

Където s - низът за сравнение с извикващия низ;

start - позиция (индекс) в низ s, от който да започне преглед на символите на низа за сравнение;

num е броят на символите в низ s за сравнение спрямо извикващия низ.

Функцията работи по следния начин. Ако извикващият низ е по-малък от низ s, тогава функцията връща -1 (отрицателна стойност). Ако извикващият низ е по-голям от низ s, функцията връща 1 (положителна стойност). Ако двата низа са равни, функцията връща 0.

Пример. Демонстрация на функцията compare ():

// тип string, функция compare()

string s1 = "012345";

string s2 = "0123456789";

int res;

res = s1.compare(s2); // res = -1

res = s1.compare("33333"); // res = -1

res = s1.compare("012345"); // res = 0

res = s1.compare("345"); // res = -1

res = s1.compare(0, 5, s2); // res = -1

res = s2.compare(0, 5, s1); // res = -1

res = s1.compare(0, 5, "012345"); // res = -1

res = s2.compare(s1); // res = 1

res = s2.compare("456"); // res = -1

res = s2.compare("000000"); // res = 1

Сравняваме низове декларирани чрез string а и чрез char b.

След като стартираме програмата трябва да въведем някакъв низ.

#include <iostream>

#include <string.h>

using namespace std;

int main()

{

string a;

cout << "Insert text: ";

cin >> a;

char b[80] = "Hello World";

cout << a << endl;

cout << "length a=" << a.length() << endl;

cout << "size a =" << sizeof(a) << endl;

cout << b << endl;

cout << "length b=" << strlen(b) << endl;

cout << "size b=" << sizeof(b) << endl;

return 0;}

**ЗАДАЧИ**

При проблеми с фунциите добавете това: #pragma warning(disable:4996)

**Задача 1.** Да се напише програма, която да намира дължината на низа „Програмиране”.

#include <iostream>

#include<windows.h>

using namespace std;

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int lenght = strlen("Програмиране");

cout << lenght << endl;

system("PAUSE");

return 0;

}

**Задача 2.** Да се напише програма, която сравнява низовете „Програмиране” и „програмиране”, „Програмиране” и „ПРОГРАМИРАНЕ”, „Програмиране” и „Програмиране”.

#include <iostream>

#include<windows.h>

using namespace std;

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int result = strcmp("Програмиране", "програмиране");

cout << result << endl;

result = strcmp("Програмиране", "ПРОГРАМИРАНЕ");

cout << result << endl;

result = strcmp("Програмиране", "Програмиране");

cout << result << endl;

system("PAUSE");

return 0;

}

**Задача 3.** Да се напише програма за въвеждане и извеждане на символни низове.

#include <iostream>

using namespace std;

int main()

{

cout << "What year was your house built? \n";

int year;

cin >> year;

cin.get();

cout << "What is its street address? \n";

char address[80];

cin.getline(address, 80);

cout << "Year built: " << year << "\n";

cout << "Address: " << address << "\n";

return 0;

}

**Задача 4.** Да се състави програма, която намира всички цифри в даден низ ( >=1) и ги замества със същите цифри намалени с единица.

*Примерен вход: adfp123076с Примерен изход*: *adfp012065с*

*Тук ползваме вградената библиотечна функция isdigit*

***#include <cctype>***

***#include <iostream>***

***#include <cstring>***

***using namespace std;***

***int main()***

***{***

***char str[] = "hj;pq910js4";***

***cout << "The digit in the string are:" << endl;***

***for (int i=0; i<strlen(str); i++)***

***{***

***if (isdigit(str[i]))***

***cout << str[i] << " ";***

***}***

***return 0;***

***}***

#include<iostream>

#include<string.h>

using namespace std;

int main() {

char s[80];

int i = 0;

cin.getline(s, 80);

while (i < strlen(s)) {

if (isdigit(s[i]) && s[i] != '0')

s[i] = s[i] - 1;

i++;

}

cout << s;

return 0;

}

**Задача 5.** Да се състави програма, която намира всички цифри в даден низ, с дължина не повече от 80 символa, и ги замества с ‘X’.

*Примерен вход: sf123sdfdf3 Примерен изход*: *sfXXXsdfdfX*

#include<iostream>

#include<string.h>

using namespace std;

int main() {

char s[80];

int i = 0;

cin.getline(s, 80);

while (i < strlen(s)) {

if (isdigit(s[i]))

s[i] = 'X';

i++;

}

cout << s;

return 0;

}

**Задача 6.** Да се изведе пълното име при зададени име и фамилия като се използва разделител интервал и функцията за конкатенация на низове.

#include <string.h>

#include <iostream>

#include <windows.h>

using namespace std;

int main(int argc, char\* argv[])

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

char firstName[20] = "Славка";

char familyName[20] = "Гайтанджиева";

char name[45] = "";

strcat(name, firstName);

strcat(name, " ");

strcat(name, familyName);

cout << name << endl;

system("PAUSE");

return 0;

}

**Задача 7.** Да се напише програма, в която са дефинирани низове със стойности ТМ и Телематика. Във вторият низ да се копира стойността на първия низ като се използва вградената функция за копиране. Да се отпечатат стойностите на низовете преди и след копирането.

#include <string.h>

#include <iostream>

#include <windows.h>

using namespace std;

int find(char seq[][15], int count, char\* pattern)

{

int i;

for (i = 0; i < count; i++)

if (!strcmp(pattern, seq[i])) return 1;

return 0;

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

char string1[] = "ТМ";

char string2[] = "Телематика";

cout << "Преди копирането: " << endl;

cout << "Низ 1:\t" << string1 << endl;

cout << "Низ 2:\t" << string2 << endl;

strcpy(string2, string1);

cout << "След копирането: " << endl;

cout << "Низ 1:\t" << string1 << endl;

cout << "Низ 2:\t" << string2 << endl;

system("PAUSE");

return 0;

}

**Задача 8.** Да се декларират 3 символни низа с дължина 15 символа за въвеждане на име, презиме и фамилия на студент.

1. Да се конкатенират имената в нов низ. Да се изведе получения низ и неговата дължина.
2. Да се изведат инициалите на студента.
3. Да се преброи колко пъти символът ‘a’ се среща в пълното име на студента

#include<iostream>

#include<windows.h>

#include<string.h>

using namespace std;

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

char name1[15];

char name2[15];

char name3[15];

char name[48];

cout << "Въведете име: ";

cin >> name1;

cout << "Въведете презиме: ";

cin >> name2;

cout << "Въведете фамилия: ";

cin >> name3;

strcpy(name, name1);

strcat(name, " ");

strcat(name, name2);

strcat(name, " ");

strcat(name, name3);

cout << name << "\n";

int n = strlen(name);

cout << "Дължина: " << n << "\n";

cout << "Инициали: " << name1[0] << '.';

cout << name2[0] << '.' << name3[0] << '.' << "\n";

int br = 0;

for (int i = 0; i <= n; i++)

if (name[i] == 'a')

br++;

cout << "Брой на 'a' = " << br << "\n";

}

**Задача 9.** Да се напише програма, която въвежда знаков низ с малки букви, преобразува ги в главни букви и извежда полученият низ. Реализирайте решение чрез функция.

#include <iostream>

using namespace std;

void main()

{

char name[80];

int loop;

cout << "Enter name in lowercase: \n";

cin.getline(name, 80);

for (loop = 0; name[loop] != 0; loop++)

name[loop] = toupper(name[loop]);

cout << "The name in uppercase is: \n" << name << endl;

system("pause");

}

**Задача 10.** Да се сравнят два въведени от потребителя низа с дължина 20 символа. Програма да копира по-големият от двата низа в трети низ и да го изведе на екрана. В случай, че двата низа са еднакви третият низ да съдържа конкатенацията на двата низа. Да се изведе съдържанието на третият низ на екрана.

#include <string.h>

#include <iostream>

#include<windows.h>

using namespace std;

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

char str1[20], str2[20], str3[40];

int result;

cout << "Въведи низ1: ";

cin >> str1;

cout << "Въведи низ2: ";

cin >> str2;

result = strcmp(str1, str2);

if (result < 0)

strcpy(str3, str1);

else

if (result > 0)

strcpy(str3, str2);

else

{

strcpy(str3, str1);

strcat(str3, str2);

};

cout << str3;

return 0;

}

**Задача 11.** Дадена е редица от n низа с дължина не по-голяма от 9. Да се напише програма, която конкатенира елементите на редицата.

#include<iostream>

#include<string.h>

using namespace std;

int main()

{

int n;

char a[20][10]; //masiv a ot 20 niza

cout << "n=";

cin >> n;

int i;

for (i = 0; i < n; i++)

{

cout << "a[" << i << "]=";

cin >> a[i];

}

char s[200] = ""; // s- rezultat ot konkatenaciqta

for (i = 0; i <= n - 1; i++)

strcat(s, a[i]);

cout << s << '\n';

return 0;

}

**Задача 12.** Да се състави програма, която сортира във възходящ ред масив от максимум 50 елемента от тип символен низ с дължина 30 символа.

#include < iostream>

#include < string.h>

#include<windows.h>

using namespace std;

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

char people[50][30];

char xname[30];

int n; // действителен брой на елементите на масива people

int i, j;

cout << "Въведете броя на елементите на масива people, n <=50 : ";

cin >> n;

for (i = 0; i < n; i++)

{

cout << "въведете people[" << i << "] = ";

cin >> people[i];

}

// сортиране на масива people

for (i = 0; i < n - 1; i++)

for (j = i + 1; j < n; j++)

{

if (strcmp(people[i], people[j]) > 0)

{

strcpy(xname, people[i]);

strcpy(people[i], people[j]);

strcpy(people[j], xname);

}

}

// Извеждане на сортирания масив people

for (i = 0; i < n; i++)

cout << people[i] << endl;

return 0;

}

**Задача 13. String**

#include <iostream>

#include <string>

using namespace std;

int main()

{

string Name;

cout << "Enter the name: ";

getline(cin, Name);

cout << "You entered name " << Name << endl;

cout << "This name is " << Name.length() << " characters long." << endl;

Name.append(" Petrov");

cout << "Appended " << Name << endl;

cout << "New name is " << Name.length() << " characters long." << endl;

for (int i = 0; i < Name.length(); i++)

cout << Name[i] << endl;

string s1, s2;

s1 = "Pexar";

s2 = "Petyr";

if (s1 < s2)

cout << s1 << " is less than " << s2 << endl;

else

cout << s1 << " is greater or equal than " << s2 << endl;

return 0;

}

**Задача 14.** Да се напише функция, която пресмята колко пъти се среща даден символ в низ.

Най-лесно задачата се решава с обхождане на низа като масив.

size\_t count\_symbol(const char\* s, const char c)

{

size\_t count = 0;

for(size\_t i=0; s[i]!='\0'; ++i)

if(s[i] == c)

++count;

return count;

}

Тази версия работи със C-style низове. В условието за край на цикъла for може да не участва нулевия байт. За целта трябва да използваме библиотечната функция strlen(). Ето как ще изглежда цикъла тогава: for(size\_t i=0; i<strlen(s); ++i)

Недостатък на такава реализация е, че на всяка итерация ще се пресмята една и съща стойност, която не зависи от операторите в тялото на цикъла. Самостоятелно променете функцията, така че strlen() да се извиква само веднъж.

Можем да решим задачата и като използваме strchr():

size\_t count\_symbol(const char\* s, const char c)

{

size\_t count = 0;

const char\* p;

while(p = strchr(s,c))

{

++count;

s = p+1;

}

return count;

}

Да разгледаме и вариант, който работи с низ от тип string.

size\_t count\_symbol(const string s, const char c)

{

size\_t count = 0;

size\_t n = s.length();

for(size\_t i=0; i<n; ++i)

if(s[i] == c)

++count;

return count;

}

Тук отново имаме поелементно обхождане на низа и сравняване на всеки символ от него с търсения. Можем да използваме и член-функцията find().

size\_t count\_symbol(const string s, const char c)

{

size\_t offset = 0;

size\_t count = 0;

while((offset = s.find(c, offset)) != string::npos)

{

++offset;

++count;

}

return count;}

**Задача 15.** Да се напише функция, която преобразува всички малки букви от символен низ в главни.

Вариант за C-style низ:

char\* upcase(char\* s)

{

for(int i=0; s[i] != '\0'; ++i)

if(s[i] >='a' && s[i] <= 'z')

s[i] = s[i] - 'a' + 'A';

return s;

}

Функцията, освен че преобразува, връща като резултат низа, който е обработила. Това позволява да я използваме в изрази, или за извеждане на екрана:

cout << upcase(str) << endl;

За по-кратък код можем да използваме библиотечната функция toupper(), която преобразува символ от малка в съответната главна буква:

char\* upcase(char\* s)

{

for(int i=0; s[i] != '\0'; ++i)

s[i] = toupper(s[i]);

return s;

}

Има и „обратна” функция – преобразуваща от главна в малка буква. Тя се нарича tolower(). Пълен списък на възможностите за преобразуване на данни можете да откриете в документацията на Visual Studio на адрес: http://msdn.microsoft.com/en-us/library/0heszx3w(VS.90).aspx

Там ще намерите и множество функции за преобразуване на числа в низове и обратно.

Да се върнем на конкретната задача. Вариантът на функция upcase(), работещ с тип string се различава по типа на резултата и параметъра – те трябва да са string&. Не можем да разчитаме също и на нулевия байт.

string& upcase(string& s)

{

size\_t n = s.length();

for(int i=0; i<n; ++i)

if(s[i] >='a' && s[i] <= 'z')

s[i] = s[i] - 'a' + 'A';

return s;}

**УПРАЖНЕНИЕ 8.2**

**ДОПЪЛНИТЕЛНИ ЗАДАЧИ:**

ЗАДАЧА 1. Да се напише приложение, в което:

**Да се дефинират функции за:**

— записване в елементите на масив от цели числа на случайни стойности от интервал, задаван чрез параметри на функцията;

— извеждане на масив от цели числа;

— възходящо сортиране на масив от цели числа по метода на мехурчето;

— низходящо сортиране на масив от цели числа по метода на пряка селекция;

— възходящо сортиране на масив от цели числа по метода на просто вмъкване.

#include <iostream>

#include <windows.h>

using namespace std;

#include <ctime>

void setArrayRandom(long a[], int L, long min, long max) {

for (int i = 0; i < L; ++i)

a[i] = rand() % (max - min + 1) + min;

}

void writeArray(const long a[], int L) {

for (int i = 0; i < L; ++i) cout << " " << a[i];

cout << endl;

}

void sortUpBubble(long a[], int L) {

for (int n = 1; n <= L; ++n)

for (int i = L - 1; 0 < i; --i)

if (a[i - 1] > a[i]) {

long temp = a[i];

a[i] = a[i - 1];

a[i - 1] = temp;

}

}

void sortDownSelection(long a[], int L) {

for (int pos = L - 1; 0 < pos; --pos) {

int iMin = pos;

for (int i = 0; i < pos; ++i)

if (a[i] < a[iMin])

iMin = i;

long temp = a[pos];

a[pos] = a[iMin];

a[iMin] = temp;

}

}

void sortUpInsertion(long a[], int L) {

for (int next = L - 2; 0 <= next; --next) {

int pos = next;

while (pos < L - 1 && a[next] > a[pos + 1]) ++pos;

long elm = a[next];

for (int i = next + 1; i <= pos; ++i) a[i - 1] = a[i];

a[pos] = elm;

}

}

int main() {

SetConsoleOutputCP(1251);

srand(time(NULL));

int const Len = 10;

long ar[Len];

setArrayRandom(ar, Len, -3, 5);

writeArray(ar, Len);

sortUpBubble(ar, Len);

writeArray(ar, Len);

sortDownSelection(ar, Len);

writeArray(ar, Len);

sortUpInsertion(ar, Len);

writeArray(ar, Len);

}

Пример 1. Този пример показва последователно и двоично търсене, реализирани чрез функции, в подреден масив. Функциите връщат *true*, точно когато намерят търсеното число в масива.

#include <iostream>

#include <windows.h>

using namespace std;

void write(long a[], int L) {

cout << "Масив: ";

for (int i = 0; i < L; ++i) cout << a[i] << " ";

cout << endl;

}

bool findSequential(const long a[], int L, long num) {

// последователно търсене по намаляване наиндексите

int i;

for (i = L - 1; 0 < i && a[i] > num; --i);

return a[i] == num;

}

bool findBinary(const long a[], int L, long num) {

// двоично търсене

int first = 0, end = L - 1, m;

while (first < end) {

m = (first + end) / 2;

// 0 <= first <= m < end <= L-1

if (a[m] < num) first = m + 1;

else end = m;

}

return a[first] == num;

}

int main() {

SetConsoleOutputCP(1251);

long ar[] = { 2, 4, 4, 6, 8 };

int const Len = sizeof(ar) / sizeof(ar[0]);

for (int L = 1; L <= Len; ++L) {

write(ar, L);

cout << " true <-- ";

for (int n = 1; n < 10; ++n)

if (findSequential(ar, L, n))

cout << n << " ";

cout << endl;

cout << " true <-- ";

for (int n = 1; n < 10; ++n)

if (findBinary(ar, L, n))

cout << n << " ";

cout << endl;

cout << " false <-- ";

for (int n = 1; n < 10; ++n)

if (!findSequential(ar, L, n))

cout << n << " ";

cout << endl;

cout << " false <-- ";

for (int n = 1; n < 10; ++n)

if (!findBinary(ar, L, n))

cout << n << " ";

cout << endl;

}

}

Пример 2. Този пример показва метода на бързата сортировка на масив.

#include <iostream>

using namespace std;

#include <ctime>

void write(long\* a, int L) {

for (int i = 0; i < L; ++i) cout << ' ' << a[i];

cout << endl;

}

void quickSort(long ar[], int Len) {

long elm = ar[0];

int Left = 0, right = Len - 1;

do {

while (elm < ar[right]) --right;

while (ar[Left] < elm) ++Left;

if (Left <= right) {

long temp = ar[Left];

ar[Left++] = ar[right];

ar[right--] = temp;

}

} while (Left < right);

if (0 < right) quickSort(ar, right + 1);

if (Left < Len - 1) quickSort(ar + Left, Len - Left);

}

int main() {

srand(time(NULL));

long ar[10];

for (int n = 1; n < 10; ++n) {

// от масива се използват само първите n елемента

for (int i = 0; i < n; ++i) ar[i] = rand() % 10;

write(ar, n);

quickSort(ar, n);

write(ar, n);

cout << endl;

}

}

ЗАДАЧА 2. Да се напише приложение, в което:

**1.** Да се дефинира функция, на която чрез параметри се съобщават два масива от цели числа, и тя връща булева стойност, показваща, дали всеки елемент на 2-я масив се съдържа в 1-я масив.

**2.** Създават се и се извеждат два масива с различни дължини от случайно избрани цели числа от интервала [5;20] и за всеки от тях се съобщава, дали всички негови елементи се съдържат в другия масив.

#include <iostream>

#include <windows.h>

using namespace std;

#include <ctime>

void writeArray(const long a[], int L) {

for (int i = 0; i < L; ++i) cout << " " << a[i];

cout << endl;

}

void setArrayRandom(long a[], int L, long min, long max) {

for (int i = 0; i < L; ++i)

a[i] = rand() % (max - min + 1) + min;

}

bool containAll(const long a1[], int L1, const long a2[], int L2)

{ // връща true, точно когато a1 съдържа всеки елемент на a2

for (int i2 = 0; i2 < L2; ++i2) {

int i1 = 0;

while (i1 < L1 && a1[i1] != a2[i2]) ++i1;

if (i1 == L1) return false;

}

return true;

}

int main() {

SetConsoleOutputCP(1251);

srand(time(NULL));

const int Len1 = 4, Len2 = 9;

long ar1[Len1], ar2[Len2];

setArrayRandom(ar1, Len1, -3, 1);

setArrayRandom(ar2, Len2, -4, 3);

cout << "1-и масив: ";

writeArray(ar1, Len1);

cout << "2-и масив: ";

writeArray(ar2, Len2);

if (containAll(ar2, Len2, ar1, Len1))

cout << "Всеки елемент на 1-я се съдържа във 2-я масив.\n";

else

cout << "Някой елемент на 1-я не се съдържа във 2-я масив.\n";

if (containAll(ar1, Len1, ar2, Len2))

cout << "Всеки елемент на 2-я се съдържа в 1-я масив.\n";

else

cout << "Някой елемент на 2-я не се съдържа в 1-я масив.\n";

}

Пример 3. Този пример показва:

* четенето на два низа, които може да съдържат интервали;
* лексикографското сравняване на два низа и извеждането на релацията между тях;
* получаването чрез библиотечна функция на дължината на низ и извеждане на съответната релация.

Обработката е показана и с тип **char \***, и с тип ***string***.

#include <iostream>

#include <windows.h>

using namespace std;

#include <cstring> // за функции за работа с низове от тип char\*

#include <string> // за функции за работа с низове от тип string

int main() {

SetConsoleOutputCP(1251);

// работа с низове от тип char\*

// тук се налага постоянен контрол на размера на наличната памет

const int maxStrLen = 200; // максимална дължина на низ

char sCh1[maxStrLen + 1], sCh2[maxStrLen + 1];

// + 1 байт за признака за край на низ

cout << "1-и низ: ";

cin.getline(sCh1, maxStrLen + 1);

// maxStrLen+1 укзава, колко е разрешената памет за запис

// cin.getline по подразбиране чете всико до знака '\n' (Enter)

// cin>> чете само до разделител (т. е. до ' ', '\t' или '\n')

cout << "2-и низ: ";

cin.getline(sCh2, maxStrLen + 2);

cout << "Низове: \"" << sCh1 << "\" ";

int rel = strcmp(sCh1, sCh2);

// strcmp(sCh1, sCh2) връща цяло число,

// резултат от лексикографкото сравняване на sCh1 и sCh2:

// положително при sCh1 > sCh2;

// нула при sCh1 == sCh2;

// отрицателно при sCh1 < sCh2.

if (rel > 0) cout << '>';

else if (rel < 0) cout << '<';

else cout << '=';

cout << " \"" << sCh2 << "\"\n";

int Len1 = strlen(sCh1), Len2 = strlen(sCh2);

cout << "Дължини: " << Len1 << ' ';

if (Len1 > Len2) cout << '>';

else if (Len1 < Len2) cout << '<';

else cout << '=';

cout << ' ' << Len2 << "\n\n";

// работа с низове от тип string

// тук НЯМА нужда от контрол на размера на наличната памет

// (той се извършва автоматично при използването на тип string)

string str1, str2;

cout << "1-и низ: ";

getline(cin, str1);

// getline(cin,str1) по подразбиране чете всико до знак '\n' (Enter)

// cin>>str1 чете само до разделител (т. е. до ' ', '\t' или '\n')

cout << "2-и низ: ";

getline(cin, str2);

cout << "Низове: \"" << str1 << "\" ";

rel = str1.compare(str2);

// str1.compare(str2) също връща цяло число,

// резултат от лексикографкото сравняване на str1 и str2:

// положително при str1 > str2;

// нула при str1 == str2;

// отрицателно при str1 < str2.

if (rel > 0) cout << '>';

else if (rel < 0) cout << '<';

else cout << '=';

cout << " \"" << sCh2 << "\"\n";

// за низове от типа string е позволено сравняване с оператор

cout << "Низове: \"" << str1 << "\" ";

if (str1 > str2) cout << '>';

else if (str1 < str2) cout << '<';

else cout << '=';

cout << " \"" << str2 << "\"\n";

Len1 = str1.length();

Len2 = str2.length();

cout << "Дължини: " << Len1 << ' ';

if (Len1 > Len2) cout << '>';

else if (Len1 < Len2) cout << '<';

else cout << '=';

cout << ' ' << Len2 << "\n\n";

}

Пример 4. Този пример показва и чрез тип **char \***, и чрез тип **string** генерирането на низ от случаен брой от 1 до 10 български букви, избирани по случаен начин от низа **„аиожхщ“.**

#include <iostream>

#include <windows.h>

using namespace std;

#include <ctime>

#include <cstring> // за функции за работа с низове от тип char\*

#include <string> // за функции за работа с низове от тип string

int main() {

#include <windows.h>

SetConsoleOutputCP(1251); // за изход

srand(time(NULL));

int Len = 1 + rand() % 10;

// с низове от тип char\*

const char bgSCh[] = "аиожхщ";

char sCh[11]; // един знак повече заради нулевия байт след низа

for (int i = 0; i < Len; ++i)

sCh[i] = bgSCh[rand() % strlen(bgSCh)];

sCh[Len] = 0;

cout << '\"' << sCh << "\"\n";

// с низове от тип string

const string bStr = "аиожхщ";

string str = "";

for (int i = 0; i < Len; ++i)

str += bStr[rand() % bStr.length()];

cout << '\"' << str << "\"\n";

}

ЗАДАЧА 3. Да се напише приложение, което въвежда низ, който може да съдържа интервали, с контрол да съдържа поне два знака и генерира и извежда низ с дължина 6, съставен само от знакове, участвуващи във въведения низ.

Да се напишат две решения, едното чрез тип **char \***, а другото чрез тип ***string***.

// 1 начин

#include <iostream>

#include <windows.h>

using namespace std;

#include <ctime>

#include <cstring> // за функции за работа с низове от тип char\*

int main() {

SetConsoleOutputCP(1251); // за изход

srand(time(NULL));

const int maxStringLen = 100;

char s[1 + maxStringLen];

do {

cout << "Низ с дължина от 2 до " << maxStringLen << ": ";

cin.getline(s, 1 + maxStringLen);

} while (strlen(s) < 2);

char newS[7];

for (int i = 0; i < 6; ++i)

newS[i] = s[rand() % strlen(s)];

newS[6] = 0;

cout << "Генериран низ: \"" << newS << "\"\n";

}

// 2 начин

#include <iostream>

#include <windows.h>

using namespace std;

#include <ctime>

#include <string> // за функции за работа с низове от тип string

int main() {

SetConsoleOutputCP(1251); // за изход

srand(time(NULL));

string s;

do {

cout << "Низ с дължина от 2 нагоре: ";

getline(cin, s);

} while (s.length() < 2);

string newS = "";

for (int i = 0; i < 6; ++i)

newS += s[rand() % s.length()];

cout << "Генериран низ: \"" << newS << "\"\n";

}

**УПРАЖНЕНИЕ 9**

**Структури: Декларация на структура. Достъп и операции над полетата на структурите с функции. Масиви от структури.**

**Структурата** е потребителски дефиниран съставен тип данни**.** Тя **представлява подредена съвкупност от ограничен брой компоненти (членове, полета), които могат да бъдат от различни типове.** Да разгледаме конкретен пример. Искаме да съхраняваме данни за книгите в една библиотека. За всяка книга е необходимо да се знае заглавие, автор, издателство и година на издаване. Ето как тази информация може да се групира в структура:

struct Book {

string title;

string author;

string publisher;

int year;

};

Ключовата дума **struct** определя Book като нов тип, от който можем да дефинираме променливи. Във фигурни скоби са изброени елементите (**полетата**) на този тип. Всеки ред представлява дефиниция на поле и завършва с точка и запетая. **Обърнете внимание, че след затварящата фигурна скоба също имаме точка и запетая.** **Това е така, защото там завършва дефиницията на структурата.** Полетата на структурата могат да бъдат от всеки тип, с изключение на същия тип структура. За нашия пример това означава, че **не** можем да имаме поле от тип Book. Това ограничение не важи по отношение на указателите, така че спокойно можем да имаме поле от тип Book\*. Дефиницията на структура не създава никакви променливи. Тя просто въвежда име на тип. Променливи от този тип се дефинират по обичайния начин:

**Book** **b**;

b е **променлива** от тип **Book** и може да съхранява данни, описани в дефиницията на структурата Book. За да зададем стойности на полетата на тази променлива ни е нужен достъп до тях.

**b.title** = "Programming Principles and Practice Using C++";

**b.author** = "Bjarne Stroustrup";

**b.publisher** = "Addison-Wesley";

**b.year** = 2009;

Полетата на структурата могат да се използват като обикновени променливи от съответния тип, например могат да участват в изрази:

int age = 2010 - b.year;

или да се въвеждат от клавиатурата:

getline(cin, b.author);

Стойности за полетата могат да се задават и при инициализация:

Book b = { "Programming Principles and Practice Using C++",

"Bjarne Stroustrup",

"Addison-Wesley",

2009 };

Виждаме, че инициализацията на структура е подобна на инициализацията на масив. Разликата е, че елементите на масива са от един и същ тип, а на структурата – от различни типове. Важно е редът на изициализиращите стойности да е същият като реда на полетата в дефиницията на структурата.

Общият вид на дефиниция на структура е следният:

**struct [<име>] {**

**<дефиниции на членове>;**

**......................**

**}[<списък от променливи>];**

В квадратни скоби са заградени незадължителните елементи на дефиницията. Виждаме, че дори името на структурата може да се пропусне. Тогава имаме **анонимна структура**. За да имаме променливи от този тип структура, трябва задължително да ги включим в списъка от променливи след затварящата фигурна скоба:

struct { //Анонимна структура

int x;

int y;

} A, B;

A.x = 10;

A.y = 20;

cout << '(' << A.x << ',' << A.y << ")\n";

Няколко последователни полета от един и същ тип могат да се обединяват в един списък от имена на полета. Структурата Book можем за запишем така:

struct Book {

string title, author, publisher;

int year;

};

Въпреки това, препоръчваме за по-добра читаемост всяко поле да се дефинира на отделен ред. Полетата на структурата се записват в паметта в реда, в който са описани в нейната дефиниция. Размерът на паметта**,** необходима за цялата

структура не винаги съвпада със сумата от размерите на всички нейни полета. Заеманата от цялата структура памет може да бъде по-голяма. Причина за това е „подравняването” на някои типове при машинното им представяне.

Операциите, които можем да извършваме с цели структури (а не с техните полета) са ограничени. Една такава операция е присвояването.

Book b2;

b2 = b;

След това присвояване всички полета на b се копират в съответните полета на b2. При това копиране може да възникне проблем, ако имаме полета, които са указатели към динамична памет. За това трябва да се избягват такива присвоявания, докато не се усвоят обектно-ориентираните техники за справяне с тези проблеми.

Две променливи от тип структура могат да съдържат еднакви данни, но не е възможно сравняване на целите структури:

if(b2 == b)cout<< "OK\n"; // Грешка!

Също така не можем да въвеждаме от клавиатурата и извеждаме на екрана цели структури:

cin >> b;

cout << b;

**Но можем да предаваме структури като параметри на функции и да връщаме структура като резултат от функция. За разлика от масивите, по подразбиране структурите се предават по стойност, а не по адрес.**

void ReadBook(Book& b)

{

cout << "Enter the title: ";

getline(cin, b.title);

cout << "Enter the author: ";

getline(cin, b.author);

cout << "Enter the publisher: ";

getline(cin, b.publisher);

cout << "Enter the year: ";

cin >> b.year;

}

void WriteBook(const Book b)

{

cout << b.title << endl;

cout << b.author << endl;

cout << b.publisher << endl;

cout << b.year << endl;

}

За да може въведените стойности да се запишат в структурата – фактически параметър, трябва да я предадем чрез псевдоним на ReadBook().

В WriteBook() структурата се предава по стойност, защото тази функция само ще изведе информацията, без да променя нищо. За по-ефективна реализация, WriteBook() може да получи своя параметър и като константен псевдоним:

void WriteBook(const Book& b)

**Както можем да очакваме, дефинирането на указател към структура не се различава от дефинирането на указател към стандартните типове:**

Book\* bp;

Такъв указател може да приема стойност адрес на променлива от тип Book.

bp = &b;

За достъп до полетата на структурата чрез указател може да се използва операцията звезда (dereferencing operator), в комбинация с операция точка:

(\*bp).year = 2010;

Скобите са от съществено значение, поради приоритета на двете операции.

Тъй като такива операции се използват често, в езика е предвидена специална операция за достъп до членове на структура (или клас) чрез указател. Тази операция се означава с -> (знак минус, последван от знак по-голямо) и се нарича операция за индиректен достъп (indirect member selection operator). С нейна помощ можем да пренапишем горния оператор така:

bp->year = 2010;

Разбира се, чрез указател можем да заделим динамична памет за структура:

Book\* bp = new Book;

ReadBook(\*bp);

//..................

delete bp;

**Няма разлика и в използването на масиви от структури.**

const int n=10;

Book collection[n];

for(int i=0; i<n; ++i)

ReadBook(collection[i]);

for(int i=0; i<n; ++i)

WriteBook(collection[i]);

За достъп до поле на структура, която е елемент на масив отново използваме операцията точка:

collection[5].year = 2010;

**Действия със структури**

* **Деклариране**
* **Инициализиране**
* **Достъп до полетата на структура**
* **Структури като аргумент и резултат на функции**
* **Указатели към структури**
* **Масиви от структури**

Пример 1: Декларираме низ, чрез служебната дума char[30] и string, забележете разликата как се присвояват стойности на a.first и a.second .

#include <iostream>

#include <string.h>

using namespace std;

struct ime

{

//string first;

char first[30];

string second;

int year;

};

int main()

{

ime a;

//a.first="Ivan";

strcpy(a.first, "Ivan");

a.second = "Asenov";

a.year = 19;

cout << a.first << "," << a.second << "," << a.year << endl;

return 0;

}

**ЗАДАЧИ**

**Задача 1.** Да се състави програма за обслужване на библиотека. За всяка книга се въвежда следната информация - шифър по каталог (цяло число), заглавие, единична цена, брой. Да се декларира масив с 30 елемента от типа на декларираната структура. Програмата да дава възможност за получаване на следните резултати:

* Извеждане на списък на всички книги с въведените данни за тях;
* Да се пресметне и изведе общата сума, която е платена за всички налични книги в библиотеката;
* Да се пресметне и изведе максималната сума, дадена за книга от библиотеката;

*Упътване: Данните за книгите се дефинират чрез структурата*

**struct book**

**{**

**int shifar;**

**char zaglavie[100];**

**double ed\_cena;**

**int br\_ekz;**

**double stoinost; //изчислява се по формулата: ед. цена \*брой екземпляри**

**};**

#include<iostream>

#include<iomanip>

#include<string.h>

using namespace std;

struct book

{

int shifar;

char zaglavie[100];

double ed\_cena;

int br\_ekz;

double stoinost;

};

void input\_data(book books[], int n1)

{

for (int i = 0; i < n1; i++)

{

cout << "Vavedi shifar: ";

cin >> books[i].shifar;

cout << "Vavedi zaglavie: ";

cin >> books[i].zaglavie;

cout << "Vavedi ed.cena: ";

cin >> books[i].ed\_cena;

cout << "Vavedi broi\_ekzemplqri: ";

cin >> books[i].br\_ekz;

books[i].stoinost = books[i].ed\_cena \* books[i].br\_ekz;

}

}

void output\_data(book books[], int n1)

{

for (int i = 0; i < n1; i++)

{

cout << left << setw(5) << books[i].shifar << " ";

cout << left << setw(20) << books[i].zaglavie << " ";

cout << left << setw(5) << books[i].ed\_cena << " ";

cout << left << setw(5) << books[i].br\_ekz << " ";

cout << left << setprecision(5) << books[i].stoinost << endl;

}

}

void obsta\_suma(book books[], int n1)

{

double sum = 0;

for (int i = 0; i < n1; i++)

{

sum = sum + books[i].stoinost;

}

cout << "Obstata suma za vsichki knigi e: " << sum;

}

void max\_data(book books[], int n1)

{

double max = books[0].stoinost;

for (int i = 1; i < n1; i++)

{

if (books[i].stoinost > max)

max = books[i].stoinost;

}

cout << "max platena suma e: " << left << setprecision(5) << max << endl;

}

int main() {

book books[30];

int n;

cout << "input broi zapisi: ";

cin >> n;

input\_data(books, n);

output\_data(books, n);

max\_data(books, n);

obsta\_suma(books, n);

return 0;

}

За домашна работа помислете: **Извеждане на списъка на всички книги, подредени по азбучен ред.**

**Задача 2.** Да се напише програма, която:

а) въвежда факултетните номера, имената и оценките по 5 предмета на студентите от една група;

б) извежда в табличен вид данните за тези студенти, чиито среден успех е по-голям или равен на 5;

**Данните за студент се дефинират чрез структурата:**

***struct student***

***{***

***int facnom;***

***char name[26];***

***int marks[5];***

***};***

#include<iostream>

#include<string.h>

#include<iomanip>

#include<stdlib.h>

using namespace std;

struct student

{

int facnom;

char name[26];

int marks[5];

};

void input\_data(student students[], int n1)

{

for (int i = 0; i < n1; i++)

{

cout << "Vavedi facnom: ";

cin >> students[i].facnom;;

cin.ignore();

cout << "Vavedi ime: ";

cin.getline(students[i].name, 26);

cout << "Vavedi 5 ocenki: ";

for (int j = 0; j < 5; j++)

cin >> students[i].marks[j];

}

}

void output\_data(student students[], int n1)

{

int sum;

for (int i = 0; i < n1; i++)

{

sum = 0;

for (int j = 0; j < 5; j++)

sum = sum + students[i].marks[j];

if ((sum / 5.0) >= 5)

{

cout << setw(5) << students[i].facnom << " ";

cout << setw(10) << students[i].name << " ";

for (int j = 0; j < 5; j++)

cout << students[i].marks[j] << " ";

cout << " sreden uspeh: " << setprecision(4) << sum / 5.0;

}

cout << endl;

}

}

int main() {

student students[30];

int n;

cout << "input broi zapisi: ";

cin >> n;

cin.ignore();

input\_data(students, n);

output\_data(students, n);

return 0;

}

**Задача 3.** Да се напише програма, която приема като вход номер, име и резултат на студент (цяло число между 2 и 6) и извежда информация за студента с най-висок резултат.

#include <iostream>

#include <iomanip>

#include<windows.h>

using namespace std;

struct Student

{

int number;

char name[20];

float result;

};

void addStudent(Student& st)

{

cout << "Номер: ";

cin >> st.number;

cout << "Име: ";

cin >> st.name;

do

{

cout << "Успех:";

cin >> st.result;

} while (st.result < 2 || st.result>6);

}

void displayStudent(Student st)

{

cout << "Номер: " << st.number << "Име: " << st.name << " Резултат: " << st.result << "\n";

}

int main()

{

SetConsoleOutputCP(1251);

Student students[20];

int bestStrudentId = 0;

int n;

cout << "Въведете брой студенти: ";

cin >> n;

for (int i = 0; i < n; i++)

{

addStudent(students[i]);

if (students[bestStrudentId].result < students[i].result)

bestStrudentId = i;

}

cout << "Студентът с най-висок резултат е: ";

displayStudent(students[bestStrudentId]);

system("pause");

return 0;

}

**Задача 4.** Да се напише програма, която приема като вход факултетен номер, име и фамилия на студент и извежда данните за студентите сортирани във възходящ ред по име.

#include <iostream>

#include<windows.h>

#include<iomanip>

using namespace std;

struct Student

{

int number;

char name[20];

char lastname[20];

};

void addStudent(Student& st)

{

cout << "Факултетен номер: ";

cin >> st.number;

cout << "Име: ";

cin >> st.name;

cout << "Фамилия:";

cin >> st.lastname;

}

void displayStudents(Student st[], int n)

{

cout << "Фак.номер" << setw(10) << "Име" << setw(10) << " Фамилия" << "\n";

for (int i = 0; i < n; i++)

{

cout << st[i].number << setw(15) << st[i].name << setw(15) << st[i].lastname << "\n";

}

}

void bubbleSort(Student st[], int n)

{

Student temp;

for (int i = 1; i <=n ; i++)

{

for (int j = 1; j <= i; j++)

{

if (strcmp(st[j - 1].name, st[j].name) > 0)

{

temp = st[j - 1];

st[j - 1] = st[j];

st[j] = temp;

}

}

}

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

Student students[40];

int n;

cout << "Въведете брой студенти: ";

cin >> n;

for (int i = 0; i < n; i++)

addStudent(students[i]);

cout << endl << "Изход преди сортиране: " << endl;

displayStudents(students, n);

cout << endl;

bubbleSort(students, n);

cout << endl << "Изход след сортиране: " << endl;

displayStudents(students, n);

system("pause");

return 0;

}

**Задача 5.** В един фитнес клуб се прави малка програма за работа с клиентите му.

Всеки клиент се описва с име, тегло, платена такса (лв) и брой посещения до момента.

Броят клиенти е между 10 и 100. Въведете информация за тези клиенти.

Изведете информация за:

1. всички клиенти;

2. клиенти, които са платили повече от 35.50 лв и са направили над 10 посещения;

3. клиентите с максимален и минимален брой посещения с тегло над 80 кг;

4. броя клиенти с тегло между 60 и 90 кг.

#include <iostream>

#include <string>

using namespace std;

const int MaxClients = 100;

// глобална дефиниция на структурата, за да е видима от функциите извън main()

struct Client {

string Name;

float Weight;

float Paid;

int Visits;

};

Client Clients[MaxClients + 1]; // правим и масива Clients глобален, MaxClients+1 елемента, за да работим с 1..MaxClients

void EnterNumberOfClients(int& n, int aMax)

{

do {

cout << "Enter the number of Clients (3.." << aMax << "): ";

cin >> n;

} while (n < 3 || n > aMax);

};

void EnterClient(Client& aClient, int id) // въвеждане на един елемент от тип Client

{

cout << "Enter information for Client " << id << endl;

cout << " Name: ";

cin.get(); // flush the cin's buffer

getline(cin, aClient.Name);

cout << " Weight: ";

cin >> aClient.Weight;

cout << " Paid: ";

cin >> aClient.Paid;

cout << " Visits: ";

cin >> aClient.Visits;

}

void EnterAllClients(Client ClientsArr[], const int size)

{

for (int i = 1; i <= size; i++)

EnterClient(ClientsArr[i], i);

}

void PrintClient(Client aClient, const int id)

{

cout << " " << id

<< ": name: " << aClient.Name

<< ", weight: " << aClient.Weight

<< ", paid: " << aClient.Paid

<< ", visits: " << aClient.Visits << endl;

}

void PrintAllClients(Client ClientsArr[], const int size)

{

for (int i = 1; i <= size; i++)

PrintClient(ClientsArr[i], i);

}

void PrintClientsByVisitsAndPayment(Client ClientsArr[], const int size, int Visits, float Payment)

{

for (int i = 1; i <= size; i++)

if (ClientsArr[i].Visits > Visits && ClientsArr[i].Paid > Payment)

PrintClient(ClientsArr[i], i);

}

void PrintMaxMinVisitsOverWeight(Client ClientsArr[], const int size, float aWeight)

{

int maxVisits = -1; int minVisits = -1; bool first = true;

for (int i = 1; i <= size; i++)

{

if (ClientsArr[i].Weight > aWeight)

{

// ако е 1-вия с по-голямо тегло от указаното - запомняме веднага

if (first)

{

first = false;

maxVisits = ClientsArr[i].Visits;

minVisits = ClientsArr[i].Visits;

}

// проверяваме посещенията дали са повече от най-голямото до момента

if (ClientsArr[i].Visits > maxVisits)

maxVisits = ClientsArr[i].Visits;

// проверяваме посещенията дали са под най-малкото до момента

if (ClientsArr[i].Visits < minVisits)

minVisits = ClientsArr[i].Visits;

}

}

if (first)

cout << "No Clients with Weight over " << aWeight << endl;

else

{

cout << "Maximum visits are " << maxVisits << " by: " << endl;

for (int i = 1; i <= size; i++)

if (ClientsArr[i].Weight > aWeight && ClientsArr[i].Visits == maxVisits)

PrintClient(ClientsArr[i], i);

cout << "Minimum visits are " << minVisits << " by: " << endl;

for (int i = 1; i <= size; i++)

if (ClientsArr[i].Weight > aWeight && ClientsArr[i].Visits == minVisits)

PrintClient(ClientsArr[i], i);

}

}

int CountBetween(Client ClientsArr[], const int size, float aWeightLow, float aWeightHigh)

{

int count = 0;

for (int i = 1; i <= size; i++)

if (ClientsArr[i].Weight >= aWeightLow && ClientsArr[i].Weight <= aWeightHigh)

count++;

return count;

}

int main()

{

int numClients;

EnterNumberOfClients(numClients, MaxClients);

EnterAllClients(Clients, numClients);

cout << "1. Information for all Clients: " << endl;

PrintAllClients(Clients, numClients);

cout << "2. Clients with more than 10 visits and paid more than 35.50:" << endl;

PrintClientsByVisitsAndPayment(Clients, numClients, 10, 35.50);

cout << "3. Clients with max/min visits from these with weight over 80:" << endl;

PrintMaxMinVisitsOverWeight(Clients, numClients, 80);

cout << "4. Number of clients with weight between 60 and 90 is: "

<< CountBetween(Clients, numClients, 60, 90) << endl;

}

**Задача 6.** В един университет се прави малка програма за работа със студенти. Всеки студент се описва с име, факултетен номер, курс (число от 1 до 4) и (средно-аритметичен) успех.

Изведете информация за:

1. всички студенти;

2. студенти, които имат успех под 3.50 от 2-ри курс;

3. студентите с максимален и минимален успех от 3-ти курс;

4. за всеки курс средните успехи.

#include <iostream>

#include <string>

using namespace std;

struct Student {

string Name;

string FacNo;

int Course;

float Uspeh;

};

void EnterStudent(Student& student)

{

cout << " Name: ";

cin.get(); // skip the buffer

getline(cin, student.Name);

cout << " Fac number: ";

getline(cin, student.FacNo);

cout << " Course: ";

cin >> student.Course;

cout << " Uspeh: ";

cin >> student.Uspeh;

}

void EnterStudents(Student Students[], const int size)

{

for (int i = 0; i < size; i++)

{

cout << "Enter information for student " << i << endl;

EnterStudent(Students[i]);

}

}

void PrintStudent(Student aStudent, int NumberBy)

{

cout << "Student " << NumberBy << ":"

<< " Name: " << aStudent.Name

<< " Fac number: " << aStudent.FacNo

<< " Course: " << aStudent.Course

<< " Uspeh: " << aStudent.Uspeh << endl;

}

void PrintAllStudents(Student Students[], const int size)

{

cout << "1. Information for all students: " << endl;

for (int i = 0; i < size; i++)

{

PrintStudent(Students[i], i);

}

}

float MinimalGradeFromCourse(Student Students[], const int size, int aCourse)

{

float minUspeh = 0;

for (int i = 0; i < size; i++)

{

if (Students[i].Course == aCourse)

{

// ако е 1-вия от този курс - запомняме

if (minUspeh == 0)

minUspeh = Students[i].Uspeh;

// проверяваме

if (Students[i].Uspeh < minUspeh)

minUspeh = Students[i].Uspeh;

}

}

return minUspeh;

}

float MaximallGradeFromCourse(Student Students[], const int size, int aCourse)

{

float maxUspeh = 0;

for (int i = 0; i < size; i++)

{

if (Students[i].Course == aCourse)

{

// ако е 1-вия от този курс - запомняме

if (maxUspeh == 0)

maxUspeh = Students[i].Uspeh;

// проверяваме

if (Students[i].Uspeh > maxUspeh)

maxUspeh = Students[i].Uspeh;

}

}

return maxUspeh;

}

float AverageGradeFromCourse(Student Students[], const int size, int aCourse)

{

float sum = 0;

int count = 0;

for (int i = 0; i < size; i++)

if (Students[i].Course == aCourse)

{

count++;

sum += Students[i].Uspeh;

}

if (count == 0)

return 0;

else

return sum / count;

}

void PrintAverageGradeFromCourse(float Average, int aCourse)

{

if (Average > 0)

cout << " Course " << aCourse << ": " << Average << endl;

else

cout << " Course " << aCourse << ": has no students " << endl;

}

int main()

{

const int n = 50;

Student Students[n];

int x; // работим с х студента

do {

cout << "Broi studenti (5..50): ";

cin >> x;

} while (x < 5 || x > 50); // приемаме само между 5 и 50

EnterStudents(Students, x);

// 1. всички студенти

PrintAllStudents(Students, x);

// 2. студенти, които имат успех под 3.50 от 2 - ри курс;

cout << "2. Students with grade less than 3.50 from 2-nd course: " << endl;

for (int i = 0; i < x; i++)

{

if (Students[i].Uspeh < 3.50 && Students[i].Course == 2)

PrintStudent(Students[i], i);

}

// 3. студентите с максимален и минимален успех от 3 - ти курс;

cout << "3.1. Students with minimal grade from 3-rd course: " << endl;

float max3, min3;

min3 = MinimalGradeFromCourse(Students, x, 3);

max3 = MaximallGradeFromCourse(Students, x, 3);

for (int i = 0; i < x; i++)

{

if (Students[i].Uspeh == min3 && Students[i].Course == 3)

PrintStudent(Students[i], i);

}

cout << "3.2. Students with maximal grade from 3-rd course: " << endl;

for (int i = 0; i < x; i++)

{

if (Students[i].Uspeh == max3 && Students[i].Course == 3)

PrintStudent(Students[i], i);

}

// 4. за всеки курс средните успехи

cout << "4. Average grades for course: " << endl;

for (int course = 1; course <= 4; course++)

PrintAverageGradeFromCourse(AverageGradeFromCourse(Students, x, course), course);

return 0;

}

**Задача 7.** В един супермаркет се прави малка програма за работа със стоките в него.

Всяка стока се описва с име, производител, цена и налично количество.

Въведете число n (5..50).

Въведете данните за n стоки. Използвайте структура и функции.

Изведете информация за:

1. всички стоки в магазина;

2. стоките с производител „Elite“ с цена по-ниска от 5.25 лв;

3. стоките с максимална цена от наличните в повече от 100 единици;

4. за даден производител средната цена;

5. стоките, подредени по нарастваща цена, по метода на мехурчето;

6. стоките, подредени по намаляваща цена, по метода на пряката селекция.

#include <iostream>

#include <string>

using namespace std;

const int MaxStocks = 50;

struct Stock {

string Name;

string Maker;

float Price;

float Amount;

};

Stock Stocks[MaxStocks + 1]; // правим и масива Stocks глобален, MaxStocks+1 елемента, за да работим с 1..MaxStocks

void EnterNumberOfStocks(int& n, int aMax)

{

do {

cout << "Enter the number of stocks (5.." << aMax << "): ";

cin >> n;

} while (n < 5 || n > aMax);

};

void EnterStock(Stock& aStock, int id) // въвеждане на един елемент от тип Stock

{

cout << "Enter information for stock " << id << endl;

cout << " Name: ";

cin.get(); // flush the cin's buffer

getline(cin, aStock.Name);

cout << " Maker: ";

getline(cin, aStock.Maker);

cout << " Price: ";

cin >> aStock.Price;

cout << " Amount: ";

cin >> aStock.Amount;

}

void EnterAllStocks(Stock StocksArr[], const int size)

{

for (int i = 1; i <= size; i++)

EnterStock(StocksArr[i], i);

}

void PrintStock(Stock aStock, const int id)

{

cout << " " << id

<< ": name: " << aStock.Name

<< ", maker: " << aStock.Maker

<< ", price: " << aStock.Price

<< ", amount: " << aStock.Amount << endl;

}

void PrintAllStocks(Stock StocksArr[], const int size)

{

for (int i = 1; i <= size; i++)

PrintStock(StocksArr[i], i);

}

void PrintStocksByMakerAndPrice(Stock StocksArr[], const int size, string aMaker, float aPrice)

{

for (int i = 1; i <= size; i++)

if (StocksArr[i].Maker == aMaker && StocksArr[i].Price < aPrice)

PrintStock(StocksArr[i], i);

}

void PrintMaximalPriceOverAmount(Stock StocksArr[], const int size, float anAmount)

{

float maxPrice = 0.0;

for (int i = 1; i <= size; i++)

{

if (StocksArr[i].Amount > anAmount)

{

// ако е 1-вата стока с по-голяма наличност от указаната - запомняме веднага

if (maxPrice == 0)

maxPrice = StocksArr[i].Price;

// проверяваме цената дали е по-голяма от най-голямата до момента

if (StocksArr[i].Price > maxPrice)

maxPrice = StocksArr[i].Price;

}

}

if (maxPrice == 0.0)

cout << "No stocks with amount over " << anAmount << endl;

else

for (int i = 1; i <= size; i++)

if (StocksArr[i].Amount > anAmount && StocksArr[i].Price == maxPrice)

PrintStock(StocksArr[i], i);

}

float AveragePriceByMaker(Stock StocksArr[], const int size, string aMaker, int& count)

{

count = 0;

float sum = 0.0;

for (int i = 1; i <= size; i++)

if (StocksArr[i].Maker == aMaker)

{

count++;

sum += StocksArr[i].Price;

}

if (count > 0) return sum / count;

else return 0;

}

void SortByPriceSelSort(Stock StocksArr[], const int size)

{

for (int i = 1; i <= size; i++)

{

int max = i;

for (int j = i; j <= size; j++)

if (StocksArr[j].Price > StocksArr[max].Price)

max = j;

swap(StocksArr[max], StocksArr[i]);

}

}

void SortByPriceBubbleSort(Stock StocksArr[], const int size)

{

for (int i = 1; i <= size; i++)

{

for (int j = size; j > i; j--)

if (StocksArr[j].Price < StocksArr[j - 1].Price)

swap(StocksArr[j], StocksArr[j - 1]);

}

}

int main()

{

int numStocks;

EnterNumberOfStocks(numStocks, MaxStocks);

EnterAllStocks(Stocks, numStocks);

cout << "1. Information for all stocks: " << endl;

PrintAllStocks(Stocks, numStocks);

cout << "2. Stocks with maker Elite and price under 5.25:" << endl;

PrintStocksByMakerAndPrice(Stocks, numStocks, "Elite", 5.25);

cout << "3. Stocks with maximal price from these with amount over 100:" << endl;

PrintMaximalPriceOverAmount(Stocks, numStocks, 100);

string theMaker;

cout << "4. Average price of stocks from maker (enter it): ";

cin.get();

getline(cin, theMaker);

int count;

float avg = AveragePriceByMaker(Stocks, numStocks, theMaker, count);

if (count == 0)

cout << " No stocks from the maker " << theMaker << endl;

else

cout << " Average price of stocks from maker " << theMaker << " is " << avg << endl;

cout << "5. Stocks, sorted by Price min->max (BubbleSort: ";

SortByPriceBubbleSort(Stocks, numStocks);

PrintAllStocks(Stocks, numStocks);

cout << "6. Stocks, sorted by Price max->min (SelectionSort: ";

SortByPriceSelSort(Stocks, numStocks);

PrintAllStocks(Stocks, numStocks);

}