# лабораторна робота №3 .

Тема. Програмування розгалужень.

Мета. Засвоїти розгалуження. Навчитись писати програми які використовують алгоритм розгалуження.

Завдання. Нижче приведено кілька задач різної складності. Кожен студент має розв’язати по дві задачі кожного рівня. Якщо розв’язано задачі лише з позначкою \* то оцінка в межах 4-6. Якщо з позначками \* та \*\* то 7-9 якщо з \* а також з \*\* та \*\*\* то 10-12.

Присилати архіви з готовими проектами. Завдання 1\*.

При заданому з клавіатури значенні *х* 

Аналіз алгоритму:

Для розв’язання цієї задачі необхідно пам’ятати, що не можна знайти квадратний корінь з від’ємного числа (зверніть увагу – вираз містить два квадратних кореня).

#include <iostream>

using namespace std;

void main()

{

double x, y;

cout << "Enter x ";

cin >> x;

if (x > 0)

{

y = sqrt((x\*x\*x) - (sqrt(x - 1)));

cout << "y = " << y << endl;

}

else cout << "Error" << endl;

system("pause");

}

# Завдання 2\*.

За рейтинговою системою оцінка визначається таким чином: якщо загальний бал учня становить не менше 92% від максимального, то виставляється оцінка 12, якщо не нижче 70%, то – оцінка 8, якщо ж не нижче 50%, то – оцінка 5, в інших випадках – оцінка 2. Визначте оцінку учня, якщо він набрав *N* балів, а максимальне значення загального балу становить *S.*

Аналіз алгоритму:

У цій задачі можна використати повну або скорочену форму команди розгалуження.

#include <iostream>

using namespace std;

void main()

{

double P, N, S;

cout << "Enter the maximum number of points ";

cin >> S;

cout << "Enter the number of student points ";

cin >> N;

P = N / S \* 100;

P >= 92 ? (P = 12):(P >= 70 ? (P = 8) : (P >= 50 ? P = 5 : P = 2));

cout << "student rating = " << P << endl;

system("pause");

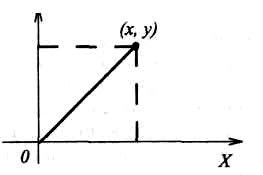
}

# Завдання 3\*\*.

На площині дано дві точки *(х1 у1 )* та *(х2, y2)*. Визначити, яка з них знаходиться далі від початку координат.

## Аналіз алгоритму:

Для розв’язання цієї задачі необхідно скористатися теоремою Піфагора для знаходження відстані від початку координат до заданої точки.



Очевидно, що відстань від початку координат до точки з координатами *(х, у)*

буде обчислюватись наступним співвідношенням:



Зверніть увагу: в зв’язку з тим, що кожна з координат у формулі підноситься до квадрату, неважливо, в якій чверті координатної площини буде знаходитись точка.

#include <iostream>

using namespace std;

void main()

{

double s1, s2, x1, x2, y1, y2;

cout << "Enter the coordinates of the first point x ";

cin >> x1;

cout << "Enter the coordinates of the first point y ";

cin >> y1;

cout << "Enter the coordinates of the second point x ";

cin >> x2;

cout << "Enter the coordinates of the second point y ";

cin >> y2;

s1 = sqrt(x1\*x1 + y1 \* y1);

s2 = sqrt(x2\*x2 + y2 \* y2);

(s1 > s2) ? (cout << "1 point further 2" << endl) : (cout << "2 point further 1" << endl);

system("pause");

}

# Завдання 4\*\*.

Чебурашка вирішив купити килими, щоб застелити кімнату, в якій він мешкав разом з Геною. Їхня прямокутна кімната виявилася розмірами *а* х *b,* де *а* та *b*

– цілі числа. Коли Чебурашка запитав у магазині, які килими є у продажу, то продавець повідомив, що є квадратні килими зі стороною *с,* де *с* – ціле число. Яку кількість килимів необхідно придбати Чебурашці, щоб максимально накрити площу кімнати? Килими не можна накладати та підгинати. Визначити, яка площа кімнати буде не накритою килимами. Передбачити ситуацію, коли розміри килиму перевищують розміри кімнати.

## Аналіз алгоритму:

Очевидно, що якщо довжина сторони килиму більша за будь-яку зі сторін кімнати, то застелити її цими килимами неможливо. Крім того, для знаходження кількості килимів, що вміщуються по одній зі сторін кімнати без їх підгинання, необхідно поділити націло довжину кімнати на довжину килиму. Загальна кількість килимів знаходиться за формулою:

*К* = *K*1 × *К*2,

де *K1* та *К2* – кількості килимів, що вміщуються вздовж двох суміжних сторін кімнати.

Площа, що не закрита килимами, визначається як різниця між площею кімнати та площею всіх куплених килимів.

Використані змінні: *a*, *b* – розміри кімнати; *с* – розмір килиму; *К1*, *К2* – кількість килимів вздовж однієї та другої стінки відповідно; *К* – загальна кількість килимів; *S* – площа кімнати, що не накрита килимами.

#include "iostream"

using namespace std;

int main()

{

int a, b, c, K, K1, K2, S, Sk, Sc;

cout << "Enter the length of the room " << endl;

cin >> a;

cout << "Enter the width of the room " << endl;

cin >> b;

cout << "Enter the length of the carpet side" << endl;

cin >> c;

if ((a > c) && (b > c)) { (K1 = a / c); (K2 = b / c); } else cout << "This carpet is too big";

K = K1 \* K2;

Sk = a \* b;

Sc = K \* (c\*c);

S = Sk - Sc;

cout << K << " carpets need to be bought" << endl;

cout << S << "m2 of the area will be uncovered" << endl;

system("pause");

}

# Завдання 5\*\*\*.

Трьом Товстунам подали на десерт кремові тістечка. Маса одного тістечка –

*Х* кг*,* а маса Товстунів відповідно *X1* кг*, Х2* кг та *Х3* кг. Перший Товстун з’їв *N* тістечок. Кожний наступний Товстун з’їдав у два рази більше від попереднього, але при цьому він не міг з’їсти більше половини своєї власної ваги. Скільки тістечок було з’їдено Товстунами за обідом?

## Аналіз алгоритму:

Зверніть увагу на те, що другий та третій Товстуни за умовою можуть з’їсти тістечок у два рази більше, ніж попередній Товстун, але не можуть з’їсти більше половини своєї ваги. Тому фактично в задачі необхідно перевірити, чи не перевищує кількість тістечок, що може з’їсти кожний Товстун, дозволену масу, і у відповідності до цього підрахувати кількість тістечок, що були з’їдені.

Наприклад, якщо другий Товстун може з’їсти 2*N* тістечок, то вага цієї їжі буде 2*N×Х* кг*.* Але за умовою він не може з’їсти більше половини своєї ваги, тобто більше ніж *X*1*/2* кг. Тому якщо вага тих тістечок, що Товстун може з’їсти, не перевищує поріг *X*1*/2* кг, то ми до загальної кількості тістечок додаємо всі можливі, тобто 2*N,* якщо ж перевищує, то ми додаємо тільки ту кількість тістечок, що не дозволяє перевищити припустимий поріг, тобто *X*1*/*2*/X* (дозволена вага їжі поділена на вагу одного "тістечка). Якщо в цьому випадку число вийде нецілим, то це означає, що Товстун з’їв тістечко не повністю.

Щоб такого не трапилось, ми робимо відкидання дробової частини після ділення.

#include "iostream"

using namespace std;

int main()

{

float x, x1, x2, x3; int n, n1, n2, n3;

cout << "Enter the weight of the cake (gram) " << endl;

cin >> x;

cout << "Enter the weight of the first fat man (kilogram)" << endl;

cin >> x1;

cout << "Enter the weight of the second fat man (kilogram)" << endl;

cin >> x2;

cout << "Enter the weight of the third fat man (kilogram)" << endl;

cin >> x3;

n1 = x1 / 2 / (x / 1000);

n2 = x2 / 2 / (x / 1000);

n3 = x3 / 2 / (x / 1000);

if ((n1 < n2) && (n1 < n3)) { n2 = n1 \* 2; n3 = n1 \* 4; }

else if ((n2 < n1) && (n2 < n3)) { n1 = n2 / 2; n3 = n2 \* 2; }

else if ((n3 < n1) && (n3 < n2)) { n1 = n3 / 4; n2 = n3 / 2; }

n = n1 + n2 + n3;

cout << "Fat men ate " << n << " cakes" << endl;

system("pause");

return 0;

}

# Завдання 6\*\*\*

Від річкового вокзалу відійшли одночасно у протилежних напрямках теплохід та турист. Теплохід рухався зі швидкістю *V*1 км/год, а турист по стежці вздовж річки зі швидкістю *V*2 км/год. Якщо через *N* годин турист передумає і вирішить попливти річкою назад за теплоходом зі швидкістю *V*3 км/год, то чи встигне він підсісти на теплохід, який має за графіком зупинку через *Y* годин після початку руху і стоїть на цій зупинці *Z* годин? Вважати на те, що всі події відбувалися протягом однієї доби.

## Аналіз алгоритму:

Якщо турист впродовж *N* годин рухався в протилежному напрямку від теплоходу, то відстань між ними в той момент, коли турист вирішив наздогнати теплохід, була наступна:

*S=(V*1*+ V*2*)\*N*

де *V*1 та *V*2 – швидкості теплоходу та туриста відповідно.

Швидкість, з якою турист почне наздоганяти теплохід, *(V*3*–V*1*)* км за годину, де *V*3 – швидкість, з якою турист попливе навздогін теплохода. Час, який буде у туриста для наздоганяння, *(Y–N + Z)* годин, тому що зупинка в теплохода буде за розкладом через *Y* годин після початку руху, але *N* годин він уже плив, а *Z* годин теплохід буде стояти на цій зупинці. Тоді за цей час турист пройде відстань:

*St=(V*3*–V*1*)×(Y–N+Z)*

Вочевидь, турист встигне підсісти на теплохід тільки в тому випадку, якщо відстань *St* буде не менше, ніж відстань, на яку теплохід перегнав туриста.

#include "iostream"

using namespace std;

void main()

{

double V1, V2, V3, N, Z, Y, St, S;

cout << "Enter the speed of the ship " << endl;

cin >> V1;

cout << "Enter the speed of the tourist " << endl;

cin >> V2;

cout << "Enter travel time for the tourist " << endl;

cin >> N;

cout << "Enter the return speed of the tourist " << endl;

cin >> V3;

cout << "Enter the time between boat stops " << endl;

cin >> Y;

cout << "Enter the stopping time of the boat " << endl;

cin >> Z;

S = (V1 + V2)\*N;

St = (V3 - V1)\*(Y - N + Z);

(St >= S) ? cout << "The tourist will have time to land on a motor ship" << endl : cout << "The tourist will not have time to get on the boat" << endl;

system("pause");

}