**Лекція 22. Особливості використання функцій. Рекурсія**

**Покажчики на функцію**

**Відкоригувати**

Синтаксис мови С++ дозволяє використовувати покажчик на функцію.

Як відомо, ім’я будь-якої функції являє собою **покажчик-константу,** що дорівнює адресі початку входження у функцію, тобто адресі її першої машинної команди. Крім констант, можна також описувати **покажчики-змінні** на функцію у вигляді:

**type (\*name) (список аргументів);**

де **type** — тип значення, що повертається функцією;**\*name** — ім’я змінної-покажчика на функцію. Покажчики на функцію потрібні в таких випадках:

* для використання як формальних аргументів у інших функціях;
* для непрямого виклику інших (резидентних) функцій (програм) початок входу в які записується у відоме місце пам’яті.

**Приклад 1.** Програмно реалізувати обчислення суми та різниці двох чисел з використанням покажчика на функцію для доступу до інших функцій.

/\* использование указателя на функцию для доступа к другим функциям — difference() і sum() \*/

#include <iostream.h>

#include <conio.h>

**int difference (int, int);** // прототип функции difference()

**int sum(int, int);**            // прототип функции sum()

**int (\*fun) (int, int);**

**void main ( )**

**{**

**int x = 20, y = 5, z;**

//присваивание указателю fun адреса функции difference()

**fun = difference;**

**z = fun(x, y);**               // вызов функции fun()

cout << "z = " << z << endl;

// присваивание указателю fun адреса функции sum()

**fun = sum;**

**z = fun(x, y);**

cout << "z = " << z << endl;

getch();

**}**

// функция определения разности двух чисел — difference()

**int difference(int a, int b)**

**{ return (a-b); }**

// функция определения суммы двух чисел — sum()

**int sum(int a, int b)**

**{ return (a + b); }**

Результати обчислень:

**z = 15**

**z = 25**

**Покажчики на функції, як і звичайні змінні, можна об’єднати в масиви.** Так, коли функції мають прототипи вигляду:

**int god (const void\*, const void \*);**

**int chena (const void\*, const void \*);**

**int nazv (const void\*, const void \*);**

**int avtor (const void\*, const void \*);**

можна описати функцію

**int (\*fcmp[4]) () {god, chena, nazv, avtor};**

У результаті буде створено масив функцій, який матиме звичайний доступ до елементів, тобто:

**int і=0;  
fcmp [і] (pt1, pt2);** — виклик функції god (pt1, pt2)

У цьому випадку, замінивши індекс, можна викликати іншу функцію.

Крім повернення результату виконання функцій у вигляді даних за значенням, можливо також здійснити повернення результату за допомогою операцій розіменування «\*» чи одержання адреси «&».

Операція розіменування «\*» означає, що функція повертає адресу на об’єкт. Функції в такому випадку з’являються як покажчики на функцію, тобто у вигляді:

**type \* fname (список формальних аргументів)**

Описані таким способом функції повинні повертати покажчик на тип (адресу), наприклад:

***Приклад 2***.

**#include <iostream.h>**

**#include <conio.h>**

**#include <stdio.h>**

**char dayweek (int data);**

**{ static char \*weekday[ ] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"} ;**

**int i;**

**char \*st ;**

**i= data % 7;**

**st = weekday [i] ;**

**puts(st);**

**return \*st; }**

Тут функція **dayweek()** одержує значення **data,** тобто число днів, що пройшли з якоїсь визначеної дати, і повертає день тижня у вигляді покажчика на **char**, оскільки **weekday** —це масив покажчиків на **char**.

**При оголошенні функції як покажчика на функцію результат можна передавати шляхом одержання адреси, тобто використовуючи символ “&”.** Така функція буде мати структуру:

**type \*funame (список формальних аргументів)**

**{ //** тіло функції**……………………**

**static type х;**

**return &x;}**

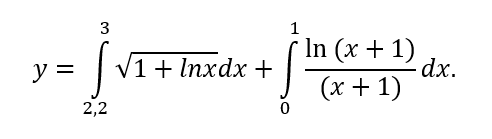
Оскільки значенням покажчика є адреса, то функція може повернути адресу об’єкта того ж типу, що і тип покажчика, який повертається. За необхідності повернення результату функції за посиланням доцільніше використовувати операцію одержання адреси “&” і функцію описувати у вигляді:

**type funame (список формальних аргументів)**

**Функції як параметр значень**

Інколи доводиться у списку формальних аргументів (параметрів) функції використовувати інші функції. Така ситуація має місце, коли при звертанні до деякої функції, треба викликати іншу функцію.

Параметр-функція записується у вигляді прототипу, тобто вказується тип функції, її ім’я і в дужках — перелік типів формальних аргументів або типів та імен формальних аргументів.

**Приклад.** Скласти програму з використанням функції обчислення інтегралів методом трапецій (точність обчислення е = 10-3)  


***Приклад 3.***

// вычисление интеграла методом трапеции

//          использование функции как параметра значения

#include <iostream.h>

#include <math.h>

#include <conio.h>

const float E = 1.e-3;

**float fn1 (float x)** // подынтегральная функция 1-го интеграла

{ return sqrt(l + log(x));}

**float fn2 (float x)** //- подынтегральная функция 2-го интеграла

{ return log(1 + pow(x,2))/(1 + pow(x,2));}

//------------------------ функция метода трапеций — ft()

**float ft (int n, float a, float b, float fun(float))**

**{ int i;**

**float s1, h, s = 0;**

do

{s1 =s;

h = (b - a)/n;

**s = (fun(a) + fun(b))/2;**

for (i = 1; і <= n-1; i++)

s += fun(a + i\*h);

s \*= h; n \*= 2; }

while (fabs(s - s1) > E);

**return s;}**

**void main ( )** //---- главная функция

**{ float y;**

**у = ft(20, 2.2, 3.0, fn1) + ft(20, 0, 1.0, fn2);**

cout << "y = " << у << endl;

getch();}

Результат виконання програми:

**у = 1.29012.**

**РЕКУРСІЯ.**

Одним зі способів опису об'єктів є рекурсія. Рекурсивними можуть бути правила, що описують структуру виразів деякої мови, означення математичних функцій, алгоритми тощо. Рекурсія є одним із фундаментальних понять програмування й математики. Завдяки їй різноманітним об'єктам можна дати зрозумілий і компактний опис.

Сукупність змінних, що утворюються під час виклику функції (підпрограми), має назву **пам'ять виклику функції**, або, не зовсім точно, – **локальна пам'ять** функції. Змінні в цій пам'яті називаються **локальними** й відповідають **параметрам** та **іменам змінних**, означеним у тілі функції. Локальна пам'ять функції містить ще один елемент – посилання на місце, з якого має виконуватися програма після закінчення виклику. Місце продовження виконання програми після виклику функції називається **точкою повернення з функції**, а посилання на неї зберігається під час виконання виклику функції.

**Поняття та приклади рекурсії**

Означення називається **рекурсивним**, якщо воно задає елементи певної множини за допомогою інших елементів цієї самої множини. Об'єкт, заданий рекурсивним означенням, також називається **рекурсивним**, а використання таких означень – **рекурсією**.

**Приклад**

1. Значення функції "факторіал" можна задати початковим елементом 0! = 1 і рекурентним співвідношенням *n*! = *n*\*(*n*-1)!.

Усі елементи цієї множини, крім першого, означаються рекурсивно. Узагалі, будь-яке рекурентне співвідношення разом із початковими умовами є прикладом рекурсивного означення.

Рекурсивне означення повинно не мати "зачарованого кола", коли в означенні об'єкта використовується він сам або інші об'єкти, задані за його допомогою. Приклад. Змінимо означення функції "факторіал": *n*! = *n*\*(*n*-1)! за *n* > 0, 0! = 1!. Значення функції від 1 виражається через її ж значення від 0, яке, у свою чергу, – через значення від 1. За цим "означенням" не можна дізнатися, яким числом є 1!.

5! = 5 \* 4 \* 3 \* 2 \* 1 = 120

4! = 4 \* 3 \* 2 \* 1 = 24

3! = 3 \* 2 \* 1 = 6

2! = 2 \* 1 = 2

1! = 1

0! = 1

Щоб обчислити факторіал, потрібно використати результати попереднього кроку

5! = 5 \* 4!

4! = 4 \* 3!

3! = 3 \* 2!

2! = 2 \* 1!

1! = 1 \* 0!

0! = 1

Варіант 1.

**int factorial (int i)**

**{**

**if (i==0)**

**return 1;**

**else**

**{**

**i=i\*factorial(i-1);**

**return i;**

**}**

}

Варіант 2.

**int factorial(int n)**

**{**

**return !n ? 1 : n \* factorial(n - 1);**

**}**

***Приклад 4***

На клавіатурі набираються цілі числа, не рівні нулю. Поява 0 означає кінець уведення. Задача: прочитати числа й видати їх у зворотному порядку (кінцевий 0 не виводити).

Перше число треба вивести останнім, друге – передостаннім і т. д. Отже, для обробки входу потрібно прочитати перше число і, якщо це не 0, то *в такий самий спосіб* обробити решту входу й потім вивести перше число. Якщо прочитано 0, то обробку вхідних даних закінчено. Ці дії описує рекурсивна функція **outReverse.**

**#include <iostream>**

**using namespace std;**

**int outReverse()**

**{ int n;**

**cin >> n;**

**if (n==0) return 0; // уведено 0 – повернення**

**outReverse(); // уведено не 0 – заглиблення**

**// після повернення з рекурсивного виклику**

**cout << n << " ";**

**return 0; }**

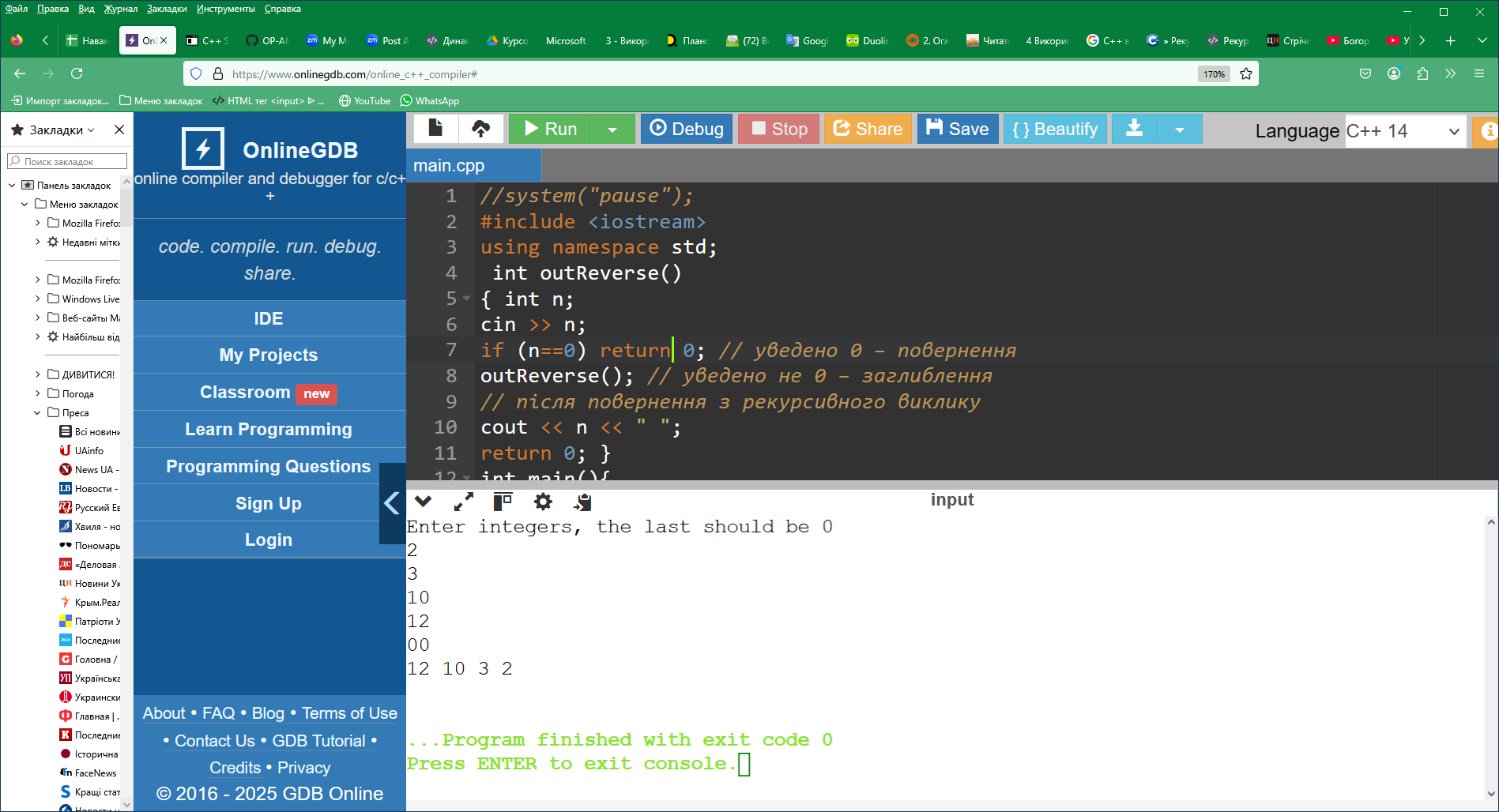
**int main(){**

**cout << "Enter integers, the last should be 0\n";**

**outReverse();**

**cout<<endl;**

**return 0; }**



З кожним рекурсивним викликом зайнята частина програмного стека збільшується, а із закінченням виклику – зменшується. Розмір стека обмежений, тому можлива ситуація (особливо за виконання рекурсивних функцій), коли пам'яті в стеку забракне й програма завершиться аварійно.

У рекурсивній функції обов'язково має бути умова, за істинності якої відбувається повернення з виклику. Ця умова визначає дно рекурсії, яке під час виконання функції обов'язково має досягатися, інакше виклики призведуть до переповнення програмного стека або інших непередбачуваних наслідків.

**Приклад 5.** Функція з *невдалою умовою* повернення з рекурсії.

**void badFunc(int x)**

**{ if(x==2) {cout << x; return;} // повертаємося**

**badFunc(x-2); // заглиблюємося }**

За виконання виклику **badFunc(6)** відбуваються рекурсивні виклики з аргументами **4** та **2**. За **x==2** заглиблення в рекурсію немає, тому виводиться **2**, а потім послідовно закінчуються виклики з аргументами **2, 4** та **6**. Проте виклик **badFunc(5)** приведе до рекурсивних викликів з аргументами **3, 1, –1, –3, …,** в яких умова повернення **x==2** ніколи не стане істинною, тому виконання рекурсивних викликів *переповнить програмний стек*.

Рекурсивна функція без жодного циклу може легко приховати величезні обсяги обчислень. Як і будь-який потужний засіб, рекурсія вимагає обережного використання.

**Глибина рекурсії й загальна кількість рекурсивних викликів**

З рекурсивними функціями пов'язано два важливих поняття – глибина рекурсії й загальна кількість викликів, породжених викликом рекурсивної підпрограми. Відрізняють глибину рекурсії, на якій перебуває виклик, і глибину рекурсії, породжену викликом.

**Глибина рекурсії, на якій перебуває виклик підпрограми** – це кількість рекурсивних викликів, розпочатих і не закінчених у момент початку цього виклику. рекурсивними функціями. Коли виконується виклик функції, який перебуває на глибині рекурсії *m*, одночасно існує *m*+1 екземпляр локальної пам'яті функції. Кожен екземпляр займає ділянку певного розміру, тому збільшення глибини може призвести до *переповнення програмного стека*.

**Загальна кількість рекурсивних викликів**, породжених викликом рекурсивної функції, – це кількість викликів, виконаних між його початком і завершенням.

Записуючи рекурсивну функцію, необхідно вміти оцінити можливу глибину рекурсії, розмір пам'яті виклику функції й загальну кількість рекурсивних викликів.

Ми розглянули *пряму рекурсію*, коли функція містить виклики самої себе.

Непряма рекурсія. Приклад.

**int firstFunction() {**

**…**

**secondFunction()**

**…**

**}**

**int secondFunction() {**

**…**

**firstFunction()**

**…**

**}**

**Швидке сортування. Приклад використання рекурсії.**

Суть полягає у розділенні масиву таким чином, щоб кожен елемент лівої частини не був більший за елемент правої частини (якщо ми говоримо про зростання).

Найпростішим способом реалізації цього алгоритму є рекурсія.

Є немало варіацій алгоритму, але головною їх відмінністю – це те, який елемент обрано опорним для перевірки.

***Приклад 6.*** Швидке сортування:

// Швидке сортування

#include <iostream>

#include <algorithm>

#include <vector>

#include <ctime>

#include <cstdlib>

using namespace std;

int Quick\_sort (int b[], int B, int E);

int main ()

{srand(time(NULL));

//system("color A");

const short int size = 20;

int a[size];

// Заповнення

cout<< "\n" << "First array: " <<endl;

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

{ a[i] = rand() % 100;

cout<< a[i] << "\t"; }

cout<<endl;

cout<<Quick\_sort(a,0, size-1);

cout<< "\n" << "After sort: " <<endl;

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

cout<< a[i] << "\t";

cout<<endl;

//system("pause");

return 0;

}

int Quick\_sort (int b[], int B, int E)

{ long i = B, j = E;

int p = b[(B+E)/2];

do{

while (b[i] < p)i++;

while (b[j] > p)j--;

if (i<=j)

{

swap(b[i], b[j]);

i++;

j--;

}

}while (i<=j);

if (B<j)return Quick\_sort(b,B,j);

if (i<E) return Quick\_sort(b,i,E);

return 0;}

Спершу потрібно вибрати опорний елемент (початок, середина, кінець…). Далі всі елементи менші за опорний перемістити ліворуч нього, а більші – праворуч. Тепер масив складається з двох частин, де елементи лівої менші за елементи правої частини. Далі повторюємо цю ж дію для обох цих частин, доки кількість елементів у них не буде меншою за 2.

**Приклад функції, яка повертає результат - покажчик.**

***Приклад 7.***

#include <iostream.h>

#include <conio.h>

#include <stdio.h>

char\* dayweek (int data)

{ static char \*weekday[ ] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"} ;

int i;

char \*st ;

i= data % 7;

st = weekday [i] ;

puts(st);

return st; }

Інший приклад

**char\* mychar()**

{ return "Hello"; }

**char\* mycharstack()**

{

char\* ch = "Hello World";

return ch;

}

**int main()**

{ cout << "mychar() = " << mychar() << endl;

cout << "mycharstack() = " << mycharstack() << endl;

cout << dayweek(30) << endl;

system("PAUSE");

return 0;

}

# Передача рядка у функцію

***Приклад 8.***

#include <iostream>

using namespace std;

//визначення функції

**void showText1** (char str[]) //Ф-ція приймає рядок як масив **4**

{ cout << str << endl; }

)

//покажчик \*str вказує на адресу першого символу в рядку

**void showText2** (char \*str) // **9**

{ cout << str << endl; }

//адрес строки из 150-ти символов

**void showText3** (char (&str)[150]) // **14**

{

cout << str << endl;

}

**int main**()

{

// Можна ввести рядок в круглих дужках при виклику функції

showText1("~~~ Example1 ~~~");

cout << endl;

// str1 - передаєм, як масив в функцію void showText1 (char str[]);"

char str1[] = " Example2";

showText1(str1); // **29**

cout << endl;

/\* str2 - передаем, в функцию void showText2 (char \*str); используя указатель.\*/

char str2[] = " Example3";

showText2(str2); // **32**

cout << endl;

/\* str3 - передаем, в функцию void showText3 (char &str[]);\nТут используем адрес строки.\*/

char str3[150] = " Example4"; // **38**

showText3(str3);

cout << endl;

system("pause");

return 0;

}

Завдання всіх наших створених функцій - відобразити на екрані рядка, які передані їм, як параметри. Підемо по порядку:

1) перший спосіб – функція void showText1 (char str[]) сприймає рядок, як масив – **рядок 4**. При визначенні функції, в круглих дужках треба вписати тип масиву (рядки) char, ім'я рядка і квадратні дужки []. Зверніть увагу, що нам не треба вказувати розмірність масиву (кількість символів в рядку). За допомогою [символи '\0′, який автоматично додається в кінець будь-якого рядка](http://cppstudio.com/uchebniki/yazyk-programmirovaniya-s/simvoly-i-stroki-v-s/), функція сама знайде останній елемент масиву і припинить висновок елементів на екран.

2) другий спосіб, передача рядка в функцію за допомогою покажчика, найпоширеніший варіант – **рядок 9**. При визначенні функції, перед ім'ям рядка ставимо оператор \* – void showText2 (char \*str).

3) ну і третій спосіб, передача в функцію рядка за адресою – **рядок 14**. У нашому прикладі це функція void showText3 (char (&str)[150]). Тут слід бути уважним, оскільки. необхідно вказувати точний розмір рядка. Без цього компілятор видасть помилку. А так само важливо взяти в круглі дужки оператор & з ім'ям рядка – (&str). В **рядку 38** нашого коду, ви можете бачити, що рядок char str3[150] містить явно менше символів. Решті, НЕ ініціалізувати, буде присвоєно значення '\0'.

Р**ядки 23-25** говорять самі за себе і коментувати не варто. Іноді, так зручно зробити, не оголошуючи і не визначаючи рядок заздалегідь.

При виклику функцій все просто - треба лише внести назву необхідної рядки в круглі дужки. **Рядки 29, 34, 39.**

***Контрольні запитання для самоперевірки***.

1. Коли використовуються покажчики на функцію?
2. Як використовується параметр-функція?
3. Що таке рекурсія? Які переваги і недоліки від її використання?.
4. Дайте визначення структури.

**Для самостійного вивчення** *(2 години)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

**Рекомендована література**

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL: <http://www.ph4s.ru/bookprogramir_1.html>
3. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
4. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL: <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>