**Лекція 6. Особливості використання функцій. Рекурсія. Використання структур.**

**Покажчики на функцію**

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()

**void main ( )**

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

**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;}**

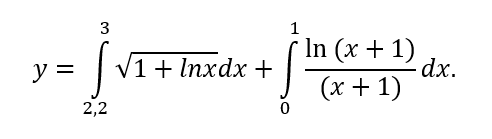
Оскільки значенням покажчика є адреса, то функція може повернути адресу об’єкта того ж типу, що і тип покажчика, який повертається. За необхідності повернення результату функції за посиланням доцільніше використовувати операцію одержання адреси “&” і функцію описувати у вигляді:

**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;**

**void outReverse()**

**{ int n;**

**cin >> n;**

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

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

**// після повернення з рекурсивного виклику**

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

**return; }**

**int main(){**

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

**outReverse();**

**cout<<endl;**

**system("pause"); 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 <time>

#include <stdlib>

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);**

**}**

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; }

**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 символа в рялку

**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 функции

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

cout << endl;

// str1 - передаем, как массив в функцию 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'.

Думаю, **рядки 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.**

# Використання структур

**Структура** — це сукупність різнотипних елементів, яким присвоюється одне ім’я (воно може бути відсутнім), що займає одну ділянку пам’яті. Елементи, що складають структуру, називаються полями.

Змінна типу структура, як і будь-яка змінна, повинна бути описана. Цей опис складається з двох кроків: опису шаблону (тобто складу) або типу структури та опису змінних структурного типу.

Синтаксис опису структури має вигляд:

**struct [<ім’я структури>]**  
**{ <тип 1> ім’я поля 1;**  
**<тип 2> ім’я поля 2 . . .;**  
**} р1, р2 . . .;**

де **struct** — службове слово;

**<ім’я структури>** — ім’я типу структура (може бути відсутнім);

**<тип 1>, <тип 2>** — імена стандартних або визначених типів;

**ім’я поля 1, ім’я поля 2,…** — імена полів структури;

**р1, р2 . . .;** — імена змінних типу структура.

Наприклад, для знаходження середнього бала, отриманого студентами в період сесії з дисциплін «Математика», «Фізика» та «Програмування», визначимо таку структуру:

**struct stud**

**{ char fam [25];**            // фамилия и инициалы

**int mat, fiz, prg;**          // предметы

**float sb;**                   // средний балл

**}** **st1, st2;**

Змінні **st1 і st2** можна оголосити окремим оператором, наприклад:

**struc stud stl st2;.**

Ініціювання полів структури слід здійснювати або при її описі, або в тілі програми. При описі структури ініціювання полів виглядає, наприклад, так:

**struct stud**

**{ char fam [25];**

**int mat,fiz, prg;**

**float sb;}**

**st1 = {"Кравченко И. С.", 4, 5, 5};**

**st2 = {"Тесленко А. М.", 3, 4, 5};**

Якщо ініціювання виконується в тілі програми, то для звернення до імені поля треба спочатку записати ім’я структурної змінної, а потім ім’я поля. Ці обидва записи відокремлюються крапкою і являють собою складене ім’я.

Отже, у випадку появи змінної **st1** у програмі для її ініцію­вання можна записати

**stud st1** **= {“Кравченко И. С.”, 4, 5, 5};** або ініціювання виконується за допомогою складених полів. Розглянемо ілюстраційну програму.

***Приклад 9.***

**#include <iostream.h>**

**#include <string.h>**

# **#include <stdio.h> // для puts: int puts(const char \*str)**

**#include <conio.h>**

using namespace std;

**main ( )**

**{ struct stud**     //----- описание структуры

**{ char fam [20];**

**int mat, fiz, prg;**

**float sb;**

**} st1, st2;**

**strcpy (st1.fam, "Кравченко И. С.");**

**st1 .mat = 4;**

**st1 .fiz = 5;**

**st1 .prg = 5;**

**st1.sb = float (st1.fiz + st1.mat + st1.prg)/3;**

**st2 = st1;**

**puts (st2.fam);**      //---------------- вывод фамилии

**cout << st2.mat << st2.fiz << st2.prg << st2.sb << endl;**

**getch();**     //задержка экрана вывода результата

**}**

У наведеній програмі організується присвоювання всім полям структури **st1** відповідних значень.

**char strcpy (s, \*st);** — виконує операцію копіювання байтів рядка **st** у рядок **s** (включаючи  **“\0”**; повертає **s**)

Структурна змінна **st2** того ж типу, що і **st1**, тому справедлива операція **st2 = st1;.**

Якщо функція використовує тільки один структурний тип, то цей тип можна оголосити без імені. Тоді раніше розглянуту структуру можна оголосити таким чином:

**struct**

**{ char fam [25];**

**int mat, fiz, prg;**

**float sb;**

**} stl, st2;**

Коли при описі структур у деякій функції або в межах видимості змінних у різних функціях є багато (але не всі) однакових полів, то їх слід об’єднати в окрему структуру. Її можна застосовувати при описі інших структур, тобто поля структури можуть самі бути типу **struct**. Це називається **вкладеністю структур** — її можна використати, наприклад, якщо треба обробляти списки студентів та викладачів університету. Студентські списки містять дані: прізвище та ініціали, дата (день, місяць, рік) народження, група та середній бал успішності, а в списках викладачів присутні такі дані: прізвище, ініціали, дата народження, кафедра, посада. У процесі обробки списку студентів і списку викладачів можна оголосити відповідно такі структури:

**struct stud**

**{ char fio [25];**

**int den, god;**

**char mes [10];**

**char grup;**

**flout sb; }**

struct prep  
{ char fio [25];  
int den, god;  
char mes [10];  
char kaf, dolg;  
}

В оголошених типах однакові поля можна об’єднати в окрему структуру і застосовувати її при описі інших типів. Поетапно це виглядає так:

* загальна структура —

**struct spd**

**{ char fio [25];**

**int den, god;**

**char mas[10]; }**

* структура для опису інформації про студентів —

**struct stud**

**{ spd dr;**

**char grup;**

**float sb}**

**st1, st2;**

* структура для опису інформації про викладачів —

**struct prep**

**{ spd dr;**

**char kaf [10];**

**char dolg [15];**

**} pr1, pr2;**

У структурах **stud і prep** для оголошення поля, що містить дані про прізвище і дату народження, використовується раніше описаний тип **spd**. Тепер до поля **fio, den, god, mes**можна звернутися, використовуючи запис **st1.dr.fio**, наприклад, при зверненні до функції введення:

**gets (st1.dr.fio);** або    **gets (pr1.dr.fio);**

Прототип функції gets: (б-ка cstdio) char \* gets( char \* string );

Отримати рядок зі стандартного потоку введення. Функція gets зчитує символи зі стандартного потоку введення до символу нового рядка n або до тих пір, поки не буде досягнутий кінець файлу EOF, після чого зберігає лічені символи в рядок типу char.

Символ нового рядка n не копіюватиметься в рядок.

Нульовий символ автоматично додається після останнього копійованого символу в string, щоб сигналізувати про кінець рядка.

Функція gets дещо відрізняється від функції fgets, по-перше повертає рядок зі стандартного потоку введення в якості аргументу, а, по-друге, кінцевий символ нового рядка не входить до її складу . на відміну від fgets. А ще, gets не дозволяє встановити обмеження на кількість зчитувальних символів, тому потрібно бути обережними з розміром масиву, на який вказує параметр str, щоб уникнути переповнення буфера.

Параметри: **string**Покажчик на масив типу char, в якому зберігається рядок.

Значення, що повертається. У разі успіху, функція повертає той же параметр, string. Якщо відбувається помилка, повертається нульовий покажчик.

Використовуйте функції ferror або feof для перевірки внутрішніх станів показників помилки або кінця файлу – EOF.

Після оголошення структурного типу змінних для роботи з їхніми полями можна застосовувати і покажчики, тоді опис структури матиме вигляд:

**struct stud**

**{ char fam [25];**

**int mat, fiz, prg;**

**float sb;**

**} st1, \*pst;**

Доступ до полів може здійснюватися двома способами:

* з використанням операції розіменування «\*», тобто  
  **gets ((\*pst).fam);    (\*pst).fiz = 5;**
* з використанням покажчика **->**, наприклад,  
  **gets (pst -> fam);      pst-> fiz = 5;** тощо.

**a->b це Розкриття посилання на структуру ("на член *b* об'єкту вказує *a*")**

Крім того, до полів змінної **st1** можна звертатися, вказуючи поля через операцію **«.»**, як це робилося раніше.

Дані типу структура можна об’єднати в масиви, для попереднього прикладу з урахуванням кількості студентів, структуру можна записати так:

**struct stud**

**{ char fam [20];**

**int mat, fiz, prg;**

**float sb;**

**} spis[15], \*sp = &spis[0];.**

У випадку, коли масив описується десь у тексті програми, тобто не саме після опису структури, його можна оголосити у вигляді: **stud spis [15];** — масив типу структура з ім’ям **stud**, що містить відповідну інформацію про групу із **15** студентів.

Доступ до елементів масиву типу структура здійснюється із застосуванням індексу або через покажчик-константу, яким є ім’я масиву, тобто одним з таких способів:

**strcpy (spis[1].fam, ” “);**

**spis[1].fiz = 5;**

або

**strcpy ((sp + 1) -> fam, ” “);**

**(sp + 1) -> fiz = 5;**

або

**strcpy ((\*(spis + 1)).fam, ” “);**

**(\*(sp + 1)).fiz = 5;.**

У останньому виразі **(\*(spis + 1)).fiz = 5;** потрібна зовнішня пара дужок, тому що **операція «.» («крапка») має пріоритет вище, ніж операція розіменування «\*».**

Поля структури можуть також бути масивами. Наприклад, у розглянутій структурі **stud** можна оцінки з різних предметів об’єднати в масив. Тоді структуру слід описати у вигляді:

**struct stud1**

**{ char fam [25];**

**int pred [3];**

**float sb**

**} spis[15],** **\*ps = &spis[0];.**

Звернення до полів здійснюватиметься одним із способів:

**((\*ps).fam)**

**(ps->pred [0]),**

наприклад,

**gets ((\*ps).fam);**  
**сіn >> ps -> pred[0] >> ps -> pred[1] >> ps -> pred[2];**

або**сіn >> ps -> \*(pred + 0) >> ps -> \*(pred + 1) >> ps -> \*(pred + 2).**

Доцільно також зазначити, що **бібліотека stdlib.h містить спеціальні функції для пошуку та сортування структурних змінних.**

Розглянемо використання структурного типу на прикладах.

**Приклад 9.** Увести в комп’ютер відомість успішності групи сту­дентів, які здали сесію з дисциплін «Математика», «Фізика» і «Програмування», та обчислити:

* середній бал кожного студента;
* середній бал групи за кожним предметом;
* вивести на екран прізвища відмінників з програмування.

Розглянемо перший варіант (**Р8\_1\_1.СРР**) реалізації постав­леної задачі, що може мати вигляд:

// Р8\_1\_ 1.СРР — обработка ведомости успеваемости группы студентов

//------------------ использование данных типа структура

**#include <iostream.h>**

**#include <string.h>**

**#include <conio.h>**

**void main()**

**{ const n = 5;**     //n — количество студентов

**int і;**

**float sm, sf, sp;**

/\* sm, sf.sp — сумма оценок группы соответственно по математике, физике, программированию \*/

**struct stud**

**{ char fam[25];** 

**int mat, fiz, prg;**

**float sb;**

**} ved[n];**       //ved[n] — массив студентов (ведомость) 

**sm = sf = sp = 0;**

//---- ввод и обработка информации о студентах и их успеваемости

**for (і = 0; і < n; i++)**

**{ cout <<"\*\*\*\*\* Введите информацию о "<< (i+1) << " студенте\n";** 

**{ cout<<<<"Введите фамилию и инициалы\n";**

**cin >> ved[i].fam;**

**cout <<"Оценки по матем., физике и программир.\n";** 

**cin >> ved[i].mat >> ved[i].fiz >> ved[i].prg;**

//----- вычисление среднего балла студента за сессию

**ved[i].sb = (float(ved[i].mat + ved[i].fiz + ved[i].prg))/3;**

//--- суммирование оценок в группе по предметам

**sm += ved[i].mat;** 

**sf += ved[i].fiz;** 

**sp += ved[i].prg;  } }**

//-- вывод результатов вычислений

**cout << ''\n\*\*\*\*\*\*\* Результаты сессии\n";**

**cout.precision(3);**

**for (і = 0; і < n; i++)**

**cout<<i+1<< " "<< ved[i].fam << " матем. = " << ved[i].mat << " физика = "<< ved[i].fiz << "програм. = " <<ved[i].prg << " ср. балл = " << ved[i].sb << "\n";**

**cout << "\n\nСредний балл группы по математике = "<< sm/n;**

**cout << "\nСредний балл группы по физике = "<< sf/n;**

**cout << "\nСредний балл группы по программированию = "<<sp/n;**

**cout << "\n\n\*\*\*\*\* Отличники по программированию: \n";**

**for (і = 0; і < n; i++)**

**if (ved[i].prg == 5) cout << ved[i].fam <<"\n";**

**getch (); }**

Результати виконання програми:

**\*\*\*\*\* Введите информацию о 1 студенте**

**Введите фамилию и инициалы**

**Адаменко Р. Л.**

**Оценки по матем., физике и программир.**

**4 4 5**

**\*\*\*\*\* Введите информацию о 2 студенте**

**Введите фамилию и инициалы**

**Бочаренко П. П.**

**Оценки но матем., физике и программир.**

**5 5 4**

**\*\*\*\*\* Введите информацию о 3 студенте**

**Введите фамилию и инициалы**

**Волошко А. А.**

**Оценки но матем., физике и программир.**

**4 4 3**

**\*\*\*\*\* Введите информацию о 4 студенте**

**Введите фамилию и инициалы**

**Долбня В. Е.**

**Оценки по матем., физике и программир.**

**5 5 5**

**\*\*\*\*\* Введите информацию о 5 студенте**

**Введите фамилию и инициалы**

**Ильченко Г. Г.**

**Оценки по матем., физике и программир.**

**3 3 4**

**\*\*\*\*\*\*\* Результаты сессия**

**Адаменко Р. Л. матем. = 4 физика = 4 програм. = 5 ср. балл = 4.33**

**Бочаренко П. П. матем. = 5 физика = 5 програм. = 4 ср. балл = 4.67**

**Волошко А. А. матем. = 4 физика = 4 прогам. = 3 ср. балл = 3.67**

**Долбня В. Е. матем. = 5 физика = 5 програм. =5 ср. балл = 5**

**Ильченко Е. Г. матем. = 3 физика = 3 програм. = 4 ср. балл = 3.33**

**Средний балл группы по математике = 4.2**

**Средний балл группы по физике = 4.2**

**Средний балл группы по программированию = 4.2**

\*\*\*\*\*\*\***Отличники по программированию**:

**Адаменко Р.Л**.

**Долбня В.Е.**

Наведемо другий варіант (**див. Р8\_1\_2.СРР**) розв’язання поставленої задачі з використанням покажчиків та структуру, яка містить поле оцінок з предметів у вигляді масиву.

//Р8\_1\_2. СРР — обработка ведомости успеваемости группы студентов использование указателей

**#include <iostream.h**>

**#include <string.h**>

**#include <conio.h>**

**void main()**

**{ const n = 5, k = 3;** /\* n — количество студентов,k — количество оценок \*/

**int і, j;**

**float sm, sf, sp;**

/\* sm, sf, sp — сумма оценок группы соответственно по математике, физике, программированию \*/

**struct stud**{ **char fam[25];**

**int ocen[3];**          // ocen[3] — массив оценок

**float sb;**

**} ved[n], \*pst =& ved[0];** //pst — указатель на массив

**sm = sf = sp = 0;**

//---- ввод и обработка информации о студентах и их успеваемости

**for (і = 0; і < n; i++, pst++)**

**{ cout << "Введите информацию о " << (i+i) << " студенте\n";**

**{ cout << "Введите фамилию и инициалы\n";**

**cin >> ((\* pst).fam);**

**cout << "Оценки по матем., физике и программир. \n";**

//------ ввод оценок и подсчет среднего балла студента

**(\*pst).sb=0;** 

**for (j = 0; j <3; j++)** 

**{ сіn >> (\*pst).ocen[j];**

**(\*pst).sb += (\*pst).ocen[j];**

**}**

**(\*pst).sb=(\*pst).sb/3**;      //средний балл студента за сессию

//-- суммирование оценок группы студентов по предметам

**sm += (\*pst).ocen[0];** 

**sf+= (\*pst).ocen[1];**

**sp += (\*pst).ocen[2];    } }**

//------------- вывод результатов вычислений

**pst -= n;** // pst=&ved[0]; — возвращение указателя на начало массива

**cout.precision(3)**;

**cout <<"\*\*\*\*\*\* Результаты сессии";**

**for (і = 0; і <n; і++, pst++)**

**cout <<і+1<<" "<<(\*pst).fam<<" матем. = "<<(\*pst).ocen[0]<<" физика = "<<(\*pst).ocen[i]<<" програм. = "<<(\*pst).ocen[2]<<" ср. балл = "<<(\*pst).sb << "\n";**

**cout << "\n\nСредний балл группы по математике = "<< sm/n;**

**cout << "\nСредний балл группы по физике = "<< sf/n;**

**cout << "\nСредний балл группы по программированию = "<< sp/n;**

**cout << "\n\n\*\*\* Отличники по программированию: \n";**

**pst -= n;**

**for (і = 0; і < n; i++, pst++)**

**if ((\*pst).ocen[2] == 5) cout << (\*pst).fam <<"\n";**

**getch (); }**

Слід нагадати, що запис виразу з покажчиком, наприклад, **(\*pst).fam**, рівнозначний запису **pst -> fam**.

В останніх версіях С++ до складу структури можна включати не тільки поля (тобто дані), але і функції обробки полів (методи). Це вдало ілюструє третій варіант (див. **Р8\_1\_3. СРР**) розв’язання даного прикладу (середній бал групи з кожного предмета не обчислювався):

// Р8\_1\_3. СРР — обработка ведомости успеваемости группы студентов

//------------ структура содержит функции обработки полей

**#include <iostream.h>**

**#include <string.h>**

**#include <conio.h>**

**void main()**

**{ const n=5;**

**int i;**

**struct Tstud**//------------ описание структуры

**{ char fam[15];**

**int mat, fiz, prg;**

**float sb()**          //----- функция подсчета среднего балла

**{ float s = float(mat+fiz+prg)/3;**

**return s;}**

**void wod()** //------- функция ввода данных

**{ cout<<" Введите фамилию\n";** 

**cin>>fam;** // фамилии вводятся без инициалов 

**cout<<"Enter ocenki\n";**

**сіn >> mat >> fiz >> prg;}**

**void vivod()** //------ функция вывода данных

**{ cout<<fam<<" mat="<<mat<<" fiz="<<fiz<" prg="<<prg;}**

**} ved[n];**

**for (і = 0; і < n; і++)** //- ввод исходных данных

**{ cout << " Введите информацию о " << (i+1) << " студенте \n";** 

**ved[i].vvod(); }**

//-- вывод результатов вычислений

**cout << "\*\*\*\*\*\*\* Результаты сессии \n";** 

**for (і = 0; і < n; i++)** 

**{ ved[i].vivod();** 

**cout.precision(3);**

**cout<<" sb="<<ved[i].sb() << endl; }**

**cout<<"\n\*\*\* Отличники по программированию: \n\n";**

**for (і = 0; і < n; i++)**

**if (ved[i].prg == 5) cout << ved[i].fam << endl;**

**getch ();  }**

Наведемо приклад, в якому ініціювання масиву типу струк­тура здійснюється під час опису, а в процесі роботи з масивом використовується поняття, що ім’я масиву є покажчиком на масив.

**Приклад 10.** Увести в комп’ютер таку інформацію про монітори: назву, термін гарантії, ціну. У програмі в рядках коментарів наведено інший спосіб запису звернення до полів структури.

/\* СРР — обработка данных типа структура, работа с именем массива как указателем на массив \*/

**#include <iostream.h>**

**#include <conio.h>**

**const int n=6;**

**void main ()**

**{**

**struct Monitor**

**{ char name[20];** //марка монитора

**int garant;** // срок гарантии

**float chena;** // цена

**}** **mon[n]= {" Samsung 757 NF ", 36, 1100.1,**

**" Sony CPD-C520K", 36, 11753.4,**

**" Hansol H95S TFT ", 24, 4260.2,**

**" Samsung 757 DFX ", 12, 1054.5,**

**" LG T710H Flatron 36, 855.,**

**" Samsung 192V TFT ", 36, 4252.2 };**

// инициализация массива mon[n] при описании

**int і, k;  cout << "\t \*\*\*\*\*\*\* Spisok monitorov \*\*\*\*\*\*\*** \**n";**

**for (і = 0; і < n; і++)** //------- вывод информации о мониторах

**{    cout<<(\*(mon + i)).name;**

// cout<< (mоn + і) -> name;

**cout<<(\*(mon + i)).garant;**

// cout<< (mоn + і) -> garant;

**cout<<(\*(mon + i)).chena<< "\n";**

// cout<< (mоn + і)-> chena<< "\n";

**}**

**getch(); }**

Результати обчислень:

**\*\*\*\*\*\*\* Spisok monitorov. \*\*\*\*\*\*\***

**Samsung 757 NF 36 1100.1**

**Sony CPD-C520K 36 11753.4**

**Hansol H95S TFT 24 4260.2**

**Samsung 757 DFX 12 1054.5**

**LG T710H Flatron 24 855**

**Samsung 192V TFT 36 4252.2**

***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.
5. Поясніть поняття "вкладеність структур".
6. Як можна звертатися до полів структури.
7. Як описується доступ до елементів масиву типу структура.
8. Як ініціалізуються 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>