**Лекція № 9.** **Символьні та рядкові величини. Складні типи даних – масиви, структури.**

**На період** карантину в дистанційній формі навчання на надані в кінці запитання потрібно надати письмові відповіді в текстовому файлі (бажано WORD), надіславши їх на адресу викладача **t.i.lumpova@gmail.com**. Файл надавати з іменем у форматі

**PTBD21<Номер лекції / практичної / лабораторної>[-<Номер завдання ЛР>][літера позначення типу роботи L – лекція, P –практична, R – лабораторна]<Прізвищеанглійською>**. Наприклад, **PTBD21L**buts.doc. Кожна відповідь оцінюється в 0,5 балів. Відповіді повинні бути не довгими і змістовними. Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності відповідей-"близнюків" відповідь буде зараховуватися першому за часом надсилання.

Тему в заголовку листа записати

**AП<Номер групи>- Лекція <Номер лекції>-<Прізвище>**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**АП<Номер групи>-Запитання-<Прізвище >**.

**Строк виконання цієї роботи 27.03.2021**

**Використання масивів**

На практиці часто виникає необхідність в обробці даних у вигляді довільного набору значень, тобто масивів. **Масив** являє собою кінцеву іменовану послідовність величин одного типу, які розрізняються за порядковим номером.

Опис масиву у програмі відрізняється від опису простої змінної наявністю після імені квадратних дужок **«[ ]»**, в яких задається кількість елементів масиву (розмірність).

**У мові C++ нумерація елементів масиву починається з 0.**

**<тип> <ім’я> [n];**

**<тип> <ім’я> [n] = {значення};**

**<тип> <ім’я> [  ] = {значення}; /\*** масив відразу, можна не вказувати його розмір \*/

Приклад:

**float m [6];  
float m [6] = {3.4, 4.5, 5.6, 6.7, 8.9, 10.3};  
float m [  ] = {3.45, 4.56, 5.67, 6.78};**

Надалі кількість елементів змінити неможливо.

Для того щоб обнулити елементи оголошеного масиву, достатньо ініціювати його перший елемент: **int mas[0]={0};**.

За замовчуванням, якщо в оголошеному масиві ініціюється тільки декілька перших елементів, то його інші елементи ініціюються нулями. Так, у випадку, коли **float mas[10]= {2.2,34.56};**, останні вісім елементів масиву одержать значення **0**.

Приклад.

**#include <iostream.h>**

**#include <math.h>**

**using namespace std;**

**int main ( )**

**{ const int n = 7;**

**float x[n], y, a(10.5);**

**int i;**

**a= 10.5;**

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

**{**

**cin >> x[i]; //введення значення**

**y = a \* x[i] \* x[i] - sin(x[i]);**

**cout << " x[" <<i<<"] ="<<x[i]<<" y = " << y << endl;**

**}**

**system("pause");**

**return(0);**

**}**

Часто застосовуються багатовимірні масиви. У них позиція елемента визначається записом декількох індексів. Найбільш розповсюджені **двовимірні масиви** або матриці.

Масив задається або списком елементів у тому порядку, и якому вони розташовані у пам’яті, або подається як масив масивів, кожний з яких поміщається в свої фігурні дужки**«{}»**. При оголошенні і одночасному ініціюванні багатовимірних масивів можна опускати кількість індексів тільки першого виміру. Якщо ініціювання не здійснюється під час оголошення масиву, то кількість індексів треба вказувати явно.

На рис. 1 зображено двовимірний масив 8х8 елементів.

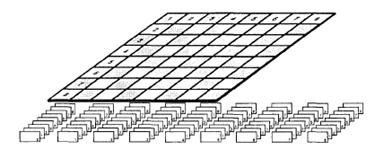


Рисунок 1 Представлення двовимірного  масиву

У програмі при описі матриці **float M[n][m];** вказується діапазон змінення двох індексів, перший з яких призначений для індексування рядків **(і)**, другий — для індексування стовпців **(j)**. Введення, обробка і виведення елементів матриці здійснюються за допомогою двох циклів, один з яких є вкладеним в іншій. Це дозволяє при кожному значенні змінної **і**перебирати всі значення змінної **j**.

**Для здійснення введення-виведення, а також для обробки елементів двовимірного масиву у програмі слід передбачати організацію двох циклів**: один — для завдання значень індексу рядків, другий — індексу стовпців.

Приклад.

**#include <iostream.h>**

**using namespace std;**

**main()**

**{**

**const int n = 3, m = 4;**

**float M [n][m], z = 10;**

**int i, j;**

**cout << "\*\*\*\* Vvod matrix " << endl;**

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

**for (j = 0; j<m; j++)**

**{ cout << " M [" << i << "]" << "[" << j << "]=";**

**cin >> M [i][j];**

**M [i][j] += z; // M [i][j]= M [i][j] + z;**

**}**

**cout << "\n\n\*\*\*\*\* Rezult matrix: ";**

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

**{ cout << endl;**

**for (j = 0; j < m; j++)**

**cout << M [i][j] << " ";}**

**system("pause");**

**}**

**Покажчики та масиви.**

**Покажчики** — це змінні, котрі містять адресу пам’яті, розподіленої для об’єкта відповідного типу. При оголошенні змінної-покажчика слід вказати тип даних, адресу яких буде містити змінна, та ім’я покажчика з символом «\*».

Загальний формат опису покажчика має вигляд:

**тип \*ім’я;**

де **тип** — тип значень, на який вказує покажчик;  
**ім’я** — ім’я змінної-покажчика;  
«\*» — операція над типом, що читається «покажчик на тип».

Наприклад:

**int \*рn** – покажчик на ціле значення;  
**float \*pf1, \*pf2;** — два покажчики на дійсні значення.

Покажчики не прив’язують дані до якого-небудь визначеного імені змінної і можуть містити адреси будь-якого неіменованого значення. Існує адресна константа **NULL**, що означає порожню адресу.

Мова C++ налічує лише дві операції, які стосуються адрес змінних, а саме:

**«&»** — **операція взяття адреси** («адреса значення»);

**«\*»** — **операція розіменування** («значення за адресою»).

Операція взяття адреси **«&»** застосовується разом зі змінною і повертає адресу цієї змінної. Операція розіменування «\*» використовується разом з покажчиками і бере значення, на яке вказує змінна-покажчик, розташована безпосередньо після символа «\*».

Оголошення покажчиків можна здійснити одним з таких способів:

**<тип> \*ptr;**  
**<тип> \*ptr = <змінна-покажчик>;**  
**<тип> \*ptr = &<ім’я змінної>;.**

Наприклад:  
**int \*ptx, b; float у;** — оголошені змінна-покажчик **ptx** та змінні **b і у**;

**float \*sp = &у;** — покажчику **sp** присвоюється адреса змінної **у**;

**float \*р = sp;** — покажчику **р** присвоюється значення (адреса значення), яке міститься в змінній **sp**, тобто адреса змінної **у**.

При оголошенні покажчиків символ «\*» може знаходитися перед ім’ям покажчика або відразу після оголошення типу покажчика і поширювати свою дію тільки на одну змінну-покажчик, перед якою він записаний:

**long \*pt;   long\*Uk;   int \*ki, x, h;** — оголошення описів.

За потреби для опису покажчика на комірку довільного типу замість ідентифікатора типу записується слово **void**, а саме:

**void \*р, \*pt;** — опис двох покажчиків на довільний тип даних.

**Перед використанням покажчика у програмі його обов’язково необхідно ініціювати**, іншими словами, необхідно присвоїти адресу якого-небудь даного, інакше можуть бути непередбачені результати.

Для одержання доступу до значення змінної, адреса якої зберігається в покажчику, досить у відповідному операторі програми записати ім’я покажчика з символом «\*» — здійснити операцію розіменування.

Розглянемо фрагмент програми з поясненнями:

**int \*р, \*р1;** — оголошені два покажчики на комірку пам’яті типу **int**;

**int х = 12, у = 5, m[7];** — оголошені змінні **х**, **у** і масив **m**, змінні ініційовані;

**\*р = &у;**     // р (&у); — покажчику **р** присвоєна адреса змінної **у**.

**cout << “Адреса р ” << р << “Значення за цією адресою = ” << \*р; ,**

Виведеться адреса комірки пам’яті, де записана змінна **у** і значення цієї змінної (**тобто 5**).

Використовуючи запис **х = \*р;**, одержимо **х = 5** тому, що **\*р = у = 5;**.

Змінити величину параметра **у** можна так:

**у = 10;**         // \*Р= 10;  
**\*р = \*р+5;**    //у +=5;.

Остання операція означає збільшення значення змінної цілого типу на **5**, тобто **у= 15**.

При ініціюванні покажчиків їм можна присвоювати або адресу об’єкта (змінної), або адресу конкретного місця пам’яті (масиву), або число 0 (нуль), а саме:

**int \*pt = (char \*) 0x00147;** — присвоюється адреса комірки;

**int \*arrpt = new int [10];** — визначається початкова адреса розміщення динамічного масиву;

**char \*р = 0;** — здійснюється ініціювання нулем.

Оскільки покажчики — це спеціальні змінні, то в операціях з іншими покажчиками вони можуть використовуватися без символа «\*», тобто без розкриття посилання, наприклад:

**float \*pt1, \*pt2, х=15, m[5];**

**pt1 = &x;**

**pt2 = pt1;**

**pt1 = m**;         //pt1 = &m[0];

де **m** — ім’я масиву, що розглядається як спеціальний покажчик-константа.

***Приклад***

**#include <iostream.h>**

**using namespace std;**

**int main ( )**

**{ int x = 10;**

**int \*px (&x); // int \*px = &x;**

**cout << "x =" << x << endl;**

**cout << "\*px =" << \*px << endl;**

**x \*= 2; //x=x\*2;**

**cout << "нове значення \*px = " << \*px << endl;\***

**px += 2; // \*px=\*px + 2;**

**cout << "результат \*px, тобто x = " << x << endl;**

**system("pause"); //затримка екрану**

**}**

Результат виконання програми:  
**х = 10**  
**\*рх = 10**  
**Новое значение \*рх = 20   
Результат \*рх, т. е. х = 22**

Для змінної-покажчика існує своя адреса і тому будуть доцільними записи:

**int \*pt1, \*pt2;**

**pt1 = (int\*) &pt2;** — покажчику **pt1** присвоюється адреса пам’ятi де розташована змінна **pt2**.

Це має сенс у випадку, коли

**int у, \*pt1, \*pt2 = &у;**

**pt1 = (int\*) &pt2;.**

**Обмеження на застосування операції взяття адреси**:

* не можна визначати адресу літеральної константи (оскільки для неї не виділяється комірка пам’яті), тобто такий запис, як **vp = &345;** — неприпустимий;
* не можна визначати адресу результату арифметичного виразу, тобто запис **vp = &(x + y);** теж неприпустимий.

**Дозволені операції для змінних-покажчиків:**

* операція розіменування «\*»;
* операція взяття адреси «&»;
* операція присвоювання «=»;
* операції інкремент «++» і декремент « –-»;
* операції додавання «+» і віднімання «-»;
* операції відношення (порівняння) покажчиків однакового типу: «==», «!=», «<», «<=», «>», «>=».

У мові C++ масиви і покажчики зв’язані між собою: ***ім’я масиву визначається як покажчик-константа на початковий (нульовий)елемент масиву.*** Так, наприклад, при оголошенні одновимірного масиву у вигляді **int mas [20];** його ім’я **mas** – покажчик на адресу початкового елемента масиву **&mas[0]**.

Існує два способи доступу до елементів масиву:

* з використанням індексу елемента масиву, наприклад, **mas[2]** або **mas[i];**
* з використанням адресного виразу, тобто виразу з покажчиком на масив, наприклад, **\*(mas + 2)** або **\*(mas + і)**.

Ім’я покажчика на масив можна записати так:

**int mas [20];**  
**int \*ptr1;**  
**ptr1 = mas;**      *//ptr1* *= &mas[0];,*

тут вирази **&mas[0] і mas** — еквівалентні.

Оскільки в комп’ютері для масивів завжди є суцільний блок комірок пам’яті, в яких розташовуються їх елементи, то адресу наступного елемента **mas[1]** можна вказати шляхом збільшення покажчика на **1**, а саме:

**р = &mas[0];**

**р++;** *//р=р  + 1;*

Таким чином, адреса **і**-го елемента визначається як **р + і**. При цьому з урахуванням типу масиву і відведеної кількості байтів для кожного його елемента автоматично виконується операція збiльшення адреси, тобто:

**адреса х[і] = адреса х[0] + i\*sizeof (тип);** .

***Для покажчиків, які посилаються на елементи масивів різних типів, результат арифметичних операцій і операцій відношення невизначений.***

До двох покажчиків **р1 і р2**, що вказують на елементи одного масиву, застосовують операції відношення: «==», «!=», «<», «<=», «>», «>=». При цьому значення покажчиків розглядаються як цілі числа, а результат порівняння дорівнює **0** ( »неправда») або **1** («істина»). Так, відношення вигляду **р1<р2** є «істина», якщо **р1** указує на більш ранній елемент, ніж **р2**. Будь-який покажчик можна порівняти з нулем.

В арифметиці з покажчиками можна використовувати адресу неіснуючого «наступного за масивом» елемента. До покажчиків можна додавати або віднімати від них цілу величину.

В обох випадках результатом операції буде покажчик на вихідний тип, значення якого на вказане число елементів більше або менше вихідного. Тобто, якщо до покажчика **р** можна додати деяку цілу величину **n**, а саме: **р + n**, то цей вираз визначає ділянку об’єкта, що займає **n**-не місце після об’єкта, на який вказує **р**, при цьому **n** автоматично збільшується на коефіцієнт, що дорівнює відповідній довжині об’єкта. Наприклад, якщо **int** займає 4 байти, то цей коефіцієнт дорівнює чотирьом.

Допускається також операція віднімання покажчиків, що вказують на елементи одного масиву. Так, якщо **р1 < р2**, то **р2 – р1 + 1** — це число елементів масиву від **р1** до **р2** включно.

**ДАНІ СИМВОЛЬНОГО ТИПУ**

Рядок являє собою масив символів, який закінчується нуль-символом. Нагадаємо, що нуль-символ має код, що дорівнює **0**, і запис у вигляді керуючої послідовності **‘\0’**. За розташуванням нуль-символу визначається фактична довжина рядка. Кількість елементів символьного масиву складається з кількості символів у рядку плюс **1**, тому що нуль-символ також є елементом масиву.

Адреса першого символу рядка може використовуватися по- різному:

* якщо рядок застосовується при ініціюванні масиву типу **char**, адреса його першого елемента стає синонімом імені масиву. Наприклад, ідентичними є такі описи масиву:
* **char st [  ] = “Слово”;**
* **char st [6] = “Слово”;**
* **char st [6] = {‘С’ ‘л’ ‘о’ ‘в’ ‘о’ ‘\0’};**

При описі символьного масиву його ім’я — не змінна, а покажчик-константа на початок рядка, тому її не можна використовувати в деяких операціях адресної арифметики. Зокрема, не можна здійснювати операцію присвоювання вигляду:

**char st [20];**

**st = “Петренко”;** — **запис неправильний**, тому що не можна змінити значення **st**.

**Виконання дій з елементами символьного масиву** здійснюється через індекси або через покажчики. Покажчики ми розглядати в цьому курсі не будемо. Для доступу до будь- якого символу рядка використовується індекс масиву **char**. Тобто, якщо описана змінна **char str [3];**, то третім елементом масиву можна скористатися, записавши: **str [2]**.

**У процесі роботи з елементами двовимірного масиву** застосовують також індекси масиву. Якщо описаний список прізвищ **char°spis°[5]°[15];**, то для використання символу масиву слід записати: **spis [і][j]**.

**Введення рядків** можна здійснювати різними способами, найбільш розповсюдженими з яких є:

* введення шляхом ініціювання при оголошенні символьних масивів:

**char st 15] = “Диск”;**  
**char st [  ] = “Диск”;**

У цьому випадку двовимірні масиви можна ініціювати по-різному, наприклад, у вигляді:

**char str [5][20] = {“Петренко И. И. “ ,“Головко С. С. “, . . . ,};**  
**char str [ ][20] = {“Петренко И. И. “, “Головко С. С. “, . . . ,};**

* використання потокового введення **сіn >>**. Здійснюється у випадку, коли рядок не містить пропусків, тому що символ пропуску є роздільником введення даних, наприклад:

**char st [5];  сіn >> st;**  
**char str [5][20];  сіn >> str [i];**

* посимвольне введення за допомогою функції **get( )**, наприклад: **get (st[i]);**
* введення за допомогою функції **cin.get**:

**cin.get (str[i], size, endl);,**

де **size** — кількість символів, що зчитуються;

* введення з використанням функції **cin.getline**:

**cin.getline (str[i], sizeof (str[i]-l));,**  
де **sizeof()** — функція визначення розміру рядка.

Виведення рядкових даних реалізується з використанням стандартного вихідного потоку **cout**:

**cout << st;**

**cout.write(st, size);** тощо.

Для потокового введення-виведення доцільно застосовувати функції **setw(w),setprecision(d), cout.width(w) і cout.precision(d).**

Введення-виведення символьних масивів можна здійснити за допомогою відповідних функцій заголовного файла **stdio.h.**, наприклад:

* для введення рядків — **gets(st); та scanf (% s,st);**
* для виведення рядків **— puts(st); і prin,tf(% s,st);.**

ранніх версіях С++ рядки розглядалися як символьні масиви. Для роботи з ними розроблено бібліотеку функцій **string.h,** що містить ефективні засоби для роботи з рядками. Згодом була розроблена стандартна бібліотека шаблонів **Standard TemplateLibrary (STL),** яка надає більш потужні засоби, об’єднані в клас **string**.

Для обробки символьних типів даних бібліотека функцій **string.h** має велику кількість вбудованих функцій:

* функції перевірки символів;
* функції перетворення символів;
* функції перевірки рядків;
* функції маніпулювання рядками.

Функції наводяться у вигляді списків, що згруповані за їх розташуванням у заголовних файлах.

**Використання рядків типу string**

**Р**аніше розглядалися питання обробки символьних даних мови С++, у тому числі і символьних рядків. Однак в останніх версіях мови С++, починаючи з С++4.5, введена стандартна бібліотека шаблонів — **Standard Template Library (STL)**, яка містить клас **string** з більш потужними засобами обробки рядків.

Для підключення цього класу до програми слід записати директиву:

**#include <string>** (без розширення ***.h***)

і підключити простір імен бібліотеки шаблонів у вигляді

**using namespace std; .**

Після цього можна оголошувати змінні типу **string:**

**string strl, str2;**

Ініціювання рядків при оголошенні виконується одним із способів:

**string st1 = "Це рядок класу string";**  
**string st2 ("Це інший рядок класу string");.**

Значення рядка **string** містить будь-який набір символів, записаний у лапках.

Для рядків типу **string** визначено такі операції:

* конкатенації (приєднання), котрі позначаються символом «+»;
* відношення («==», «!=», «>», «>=», «<», «<=»).

Наприклад, фрагмент  
**string st1 = "Приклад";**  
**string st2 = "рядка";**  
**string st3 = st1+’ ‘+st2;**  
**cout << st3 << endl;**  
дозволить вивести на екран повідомлення: **Приклад рядка.**

Для введення рядків **string**, крім операторів присвоювання, застосовують оператори введення даних:  
**сіn >> st;**  
**cin.getline(st, siseof(st));**  
**getline(cin,st, ‘\n’); тощо.**

Виведення рядків на екран здійснюється шляхом використання звичайних операторів виведення даних.

Рядки можна об’єднувати у масиви, які оголошуються звичайним засобом, тобто

**string sp[10];** — оголошення масиву, що містить 10 рядків.

Доступ до символів рядка здійснюється шляхом запису порядкового номера символу —індексу, який починається з нуля. Індекси можна записувати як у квадратних дужках, так і у звичайних, круглих.

Наприклад, якщо записати  
**string str="Mій рядок";,**  
то **str[2]** — це буде літера **‘й’.**

Для масивів рядків потрібний символ визначається шляхом запису двох індексів: індексу елемента масиву та індексу символу в цьому елементі, тобто у вигляді **mas[i][j].**

**Оголошення** **покажчика** на рядок здійснюється так:  
**string \*pst;**

Пам’ять для покажчика може бути виділена з будь-яким початковим значенням за допомогою функції **new**, наприклад:  
**string \*pstr1 = new string;** — оголошується порожній рядок,  
**string \*pstr2 = new string (“Новая строка”);** — покажчик вказує на рядок «Новаястрока».

Раніше оголошеному покажчику **\*pst,** який ні на що не вказує, можна присвоїти значення у вигляді

**string \*pst;**  
**pst = new string ("Это первая строка");.**

Існує багато функцій для обробки рядків типу **string,** розглянемо деякі з них.

**Функції для обробки рядків типу string**

Існує багато функцій для обробки рядків типу **string,** розглянемо деякі з них.

**string str;**

***Функції визначення довжини рядка:***

* **str.size();**
* **str.length();**
* **str.max\_size(); .**

Наприклад:

*//--------------------------- определение длины строки*

**#include <iostream.h>**

**#include <string>**

**using namespace std;**

**main()**

**{ string str, st ="И вот сама пришла волшебница зима!";**

**str =" волшебница";**

**cout << "Длина строки str = " << str.size() << "Длина строки st = " << st.length() << endl;**

**}**

Результат виконання:  
**Длина строки str =11 Длина строки st = 34**

***Функції додавання одного рядка або його частини до іншого рядка:***

* **str.append(st);** — додає рядок **st** до кінця рядка **srt;**
* **str.append(st,k,n);** — додає до рядка str n символів рядка **st,** починаючи з позиції **k.**

Наприклад:

*//--------------- додавання рядка*

**string str, st = "Substring in text";**

**str.append(st,3,6);**

**cout << "str = "<<str << endl;**

Результат виконання:  
**str = string**

Подальший розгляд функцій рядкових і символьних змінних продовжимо на наступньому занятті.

**Визначення та 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:

**#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);**

**Пояснення.**

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 доки, 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.

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

*Література*

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Єжова Л. Ф. Алгоритмізація і програмування процедур обробки інформації: Навч.-метод. посіб. для самост. вивч. дисц. К.: КНЕУ, 2000.
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..

*Контрольні запитання для самоперевірки*.

1. Що таке масив та які існують різновиди масивів?
2. Як здійснюється звернення до елементів масивів?
3. Що таке рядки та значення елементів символьного типу?
4. Що являє собою масив символьного типу?
5. Як здійснюється введення символьних даних?
6. Як виконується порівняння даних символьного типу?
7. Як визначити кількість символів у рядку?
8. Як описуються дані типу структура?
9. Які типи полів може містити структура?

*Контрольні запитання для надання письмових відповідей*.

1. Наведіть приклад використання операції конкатенації.
2. Наведіть перелік функцій С++ для введення-виведення елементів масиву?
3. Поясніть особливості використання покажчиків для роботи з масивами.
4. Наведіть приклади використання структур.