Тема: Складені типи даних. Покажчики (6 год.)

*План*

1. Покажчики
	1. Використання покажчиків як засіб збереження адреси
	2. Звернення до даних через покажчики
2. Для чого потрібні покажчики

2.1 Пам’ять стекова і динамічно розподільна

2.2 Використання оператора new для створення динамічних масивів

2.3 Використання оператора new для створення динамічних структур

Покажчики є могутнім засобом непрямого доступу до даних. Кожна змінна має адресу, доступитись до якої можна за допомогою оператора визначення адреси (&). Для збереження адреси використовуються покажчики.

Для оголошення покажчика досить установити тип об'єкта, адресу якого він буде містити, а потім ввести символ "\*" і ім'я покажчика. Після оголошення покажчик слід ініціалізувати. Якщо адреса об'єкта невідома, покажчик ініціалізуються значенням 0.

Для доступу до значення, яке записане за адресою в покажчику, використовується оператор розіменування (\*). Покажчик можна оголошувати константою. У цьому випадку не допускається присвоєння даному покажчикові нової адреси. Покажчик, що зберігає адресу константного об'єкта, не може використовуватися для зміни цього об'єкта.

Щоб виділити пам'ять для збереження якого-небудь об'єкта, використовується оператор new, а потім отримана адреса присвоюється покажчикові. Для звільнення зарезервованої пам'яті використовується оператор delete. Сам покажчик при звільненні пам'яті не знищується, тому звільненому покажчикові необхідно присвоїти нульове значення, щоб убезпечити його.

*I Покажчики*

Можливість безпосереднього доступу до пам'яті за допомогою покажчиків є одним з найбільш могутніх засобів мови програмування на С++.

При роботі з покажчиками програмісти часом зіштовхуються з досить специфічними проблемами, оскільки в деяких ситуаціях механізм роботи покажчиків може виявитися досить заплутаним. Крім того, у ряді випадків не можна однозначно відповісти на запитання про необхідність застосування покажчиків. Спробуємо послідовно, крок за кроком, освоїти основні принципи роботи з покажчиками. Однак усвідомити всю силу цих засобів можна тільки з часом.

Що ж таке покажчик?

*Покажчик* — це змінна, у якій записаний адрес комірки пам'яті комп'ютера.

Щоб зрозуміти, як працюють покажчики, необхідно хоча б в загальних рисах, ознайомитися з базовими принципами організації машинної пам'яті. Машинна пам'ять складається з послідовності пронумерованих комірок. Значення кожної змінної зберігається в окремій комірці пам'яті, яка називається її адресою. Зобразимо структуру розміщення в пам'яті чотирьох байтового цілого значення змінної theAge.

*Пам’ять*

*theAge*

10110101

01110110

11110110

11101110

101 102 103 104 105 106 107 108 109

Кожна комірка займає один байт.

 Unsigned long int theAge = 4 byte = 32 bit

 Ім’я змінної вказує на перший байт

 Адреса змінної theAge = 101

Для різних комп'ютерів характерні різні правила адресації пам'яті, що мають свої особливості. Однак у більшості випадків програмістові не обов'язково знати точну адресу змінної — цю задачу виконує комп'ютер. При необхідності таку інформацію можна одержати за допомогою оператора адреси (&). Приклад використання цього оператора приведений у програмі 1.

*Програма № 1.*

*// Приклад використання оператору визначення адреси*

#include <iostream>

using namespace std;

int main()

{

 unsigned short shortVar=5;

 unsigned long longVar=65535;

 long sVar=-65535;

 cout<<"shortVar: "<<shortVar;

 cout<<" Address of shortVar: "<<&shortVar<<"\n\n";

 cout<<"longVar: "<<longVar;

 cout<<" Address of longVar: "<<&longVar<<"\n\n";

 cout<<"sVar: "<<sVar;

 cout<<" Address of sVar: "<<&sVar<<"\n\n";

 system(“pause”);

 return 0;

}

*Тестування програми*

shortVar: 5 Address of shortVar: 0012FF7C

longVar: 65535 Address of longVar: 0012FF78

sVar: -65535 Address of sVar: 0012FF74

Press any key to continue

*Аналіз програмного коду*

На початку програми оголошуються і ініціалізуються три змінні: змінна типу unsigned short, unsigned long, long.

Потім виводяться значення й адреси цих змінних, отриманих за допомогою оператора визначення адреси (&).

Адреса розміщення змінної вибирається комп'ютером і може змінюватися при кожному наступному запуску програми. Тому результати можуть відрізнятися від приведених.

У більшості випадків нам не прийдеться безпосередньо маніпулювати адресами змінних. Важливо лише знати, який обсяг пам'яті займає змінна і як одержати її адресу в разі потреби. Програміст лише вказує компіляторові обсяг пам'яті, доступний для розміщення статичних змінних, після чого розміщення змінної по визначеній адресі буде виконуватися автоматично. Зазвичай тип long має чотирьохбайтове представлення. Це означає, що для збереження змінної цього типу потрібно чотири байти машинної пам’яті.

***1.1 Використання покажчиків як засіб збереження адреси.***

Кожна змінна програми має свою адресу, для збереження якої можна використовувати покажчик на цю змінну. Причому саме значення адреси знати не обов'язково.

Припустимо, що змінна howOld має тип int. Щоб оголосити покажчик pAge для збереження адреси цієї змінної, потрібно набрати наступний фрагмент коду:

int \* pAge=0;

Цим рядком змінна pAge оголошується покажчиком на тип int. Це означає, що pAge буде містити адресу значення типу int.

Відзначимо, що pAge нічим не відрізняється від будь-якої іншої змінної. При оголошенні змінної цілочислового типу (наприклад, int.) ми вказуємо на те, що в ній буде зберігатися ціле число. Коли ж змінна оголошується покажчиком на який-небудь тип, це означає, що вона буде зберігати адресу змінної даного типу. *Таким чином, покажчики є просто окремим типом перемінних*.

У даному прикладі змінна pAge ініціалізуються нульовим значенням. Покажчики, значення яких рівні 0, називають *пустими*. Після оголошення покажчикові обов'язково повинне присвоюватися яке-небудь значення. Якщо заздалегідь не відомо, яка адреса повинна зберігатися в покажчику, йому присвоюється значення 0. Неініціалізовані покажчики надалі можуть стати причиною великих неприємностей.

Оскільки при оголошенні покажчикові pAge було присвоєно значення 0, далі йому потрібно присвоїти адресу якої-небудь змінної, наприклад howOld. Як це зробити, показано нижче:

unsigned short int howOld=50;

unsigned short int \*pAge=0;

pAge=&howOlg;

У першому рядку оголошена змінна howOld типу unsigned short int і їй присвоєно значення 50. В другому рядку оголошений покажчик pAge на тип unsigned short int, якому присвоєно значення 0. Символ "зірочка" (\*) вказує на те, що описана змінна є покажчиком.

В останньому рядку покажчикові pAge присвоюється адреса змінної &howOlg. На це вказує оператор адреси (&) перед ім'ям змінної howOlg. Якби цього оператора не було, присвоювалася б не адреса, а значення змінної.

У нашому випадку значенням покажчика pAge буде адреса змінної howOlg, значення якої дорівнює 50. Два останні рядки розглянутого фрагмента програми можна об'єднати в одну:

unsigned short int howOld = 50;

unsigned short int \*pAge = &howOld;

Тепер покажчик pAge містить адресe змінної howOld. За допомогою цього покажчика можна одержати і значення змінної, на яку він вказує. У нашому прикладі це значення дорівнює 50. Звертання до значення howOld за допомогою покажчика pAge називається операцією розіменування або непрямого звертання, оскільки здійснюється неявне звертання до змінної howOld, адреса якої міститься у покажчику.

Непряме звертання має на увазі одержання значення змінної, адреса якої міститься в покажчику, а оператор розіменування дозволяє витягти це значення.

Оскільки покажчики є звичайними змінними, називати їх можна будь-якими коректними для змінних іменами. Для виділення покажчиків серед інших змінних багато програмістів використовують перед їх іменами символ "р" (від англ, pointer), наприклад pAge або pNumber.

Оператор непрямого звертання (або *оператор розіменування*) дозволяє одержати значення, що зберігається за адресою, яка записана у покажчику.

На відміну від покажчика, при звертанні до звичайної змінної здійснюється доступ безпосередньо до її значення. Наприклад, щоб оголосити нову змінну типу unsigned short int, а потім присвоїти їй значення іншої змінної, можна написати наступне:

unsigned short int yourAge;

yourAge = howOld;

При непрямому доступі буде отримане значення, що зберігається по зазначеній адресі. Щоб присвоїти новій змінній yourAge значення howOld, використовуючи покажчик pAge, що містить її адрес, напишіть наступне:

unsigned short int howOld = 50;

unsigned short int \*pAge = &howOld;

unsigned short int yourAge;

yourAge = \* pAge;

Оператор розіменування (\*) перед змінною pAge може розглядатися як "значення, що зберігається за адресою". Таким чином, вся операція присвоєння означає: "одержати значення, що зберігається за адресою, записаною в pAge, і присвоїти його змінній yourAge ".

*Отже, оператор розіменування можна використовувати з покажчиками двома способами: для оголошення покажчика і для його розіменування. У випадку оголошення покажчика символ зірочка ( \* ) сигналізує компілятору, що це непроста змінна, а покажчик, наприклад:*

unsigned short int \*pAge=0; // оголошується покажчик на змінну типу

// unsigned short int

*У випадку розіменування покажчика символ зірочка (\*) означає, що операція повинно здійснюватись не над самою адресою, а над значенням, яке зберігається за адресою, яка зберігається у покажчику:*

\*pAge = 5;

Також не слід путати оператор розіменування з оператором множення (\*). Компілятор по контексту визначає, який саме оператор використовується в даному випадку (поліморфізм операцій)

***1.2 Звернення до даних через покажчики.***

Розглянемо приклад програми, щоб краще збагнути різницю між покажчиком, адресою, яка зберігається в покажчику і значенням, яке записане за даною адресою.

int theVariable = 5;

int \* pPointer = &theVariable;

В першому рядку оголошено змінну цілого типу і присвоєно значення 5. В другому рядку оголошено покажчик на тип int, якому присвоєно адресу змінної theVariable. Змінна pPointer являється покажчиком і містить адресу змінної theVariable. Значення, яке зберігається за адресою, записаною в pPointer, дорівнює значенню 5.

Після того як покажчику присвоєно адресу деякої змінної, його можна використовувати для роботи з значенням цієї змінної.

*Програма № 2.*

*// Приклад використання покажчиків*

#include <iostream>

using namespace std;

int main()

{

//Оголошення змінної myAge на тип unsigned short int

 unsigned short int myAge = 5;

 unsigned short int \*pAge= &myAge; // покажчик

// Вивести на екран значення змінної і значення розіменованого покажчика.

 cout<<"myAge: "<<myAge<<"\n";

 cout<<"\*pAge: "<<\*pAge<<"\n\n";

// Змінній, адреса якої записана в pAge присвоюється значення 7.

 \*pAge=7;

//Після виконання такої операції змінна myAge буде містити значення 7.

 cout<<"myAge: "<<myAge<<"\n";

 cout<<"\*pAge: "<<\*pAge<<"\n\n";

// Змінній myAge присвоюється значення 9.

 myAge=9;

//Виводимо на екран безпосередньо через змінну myAge і шляхом

// розіменування покажчика на неї.

 cout<<"myAge: "<<myAge<<"\n";

 cout<<"\*pAge: "<<\*pAge<<"\n\n"; cin.get();

 return 0; }

*Тестування програми*

myAge: 5

\*pAge: 5

myAge: 7

\*pAge: 7

myAge: 9

\*pAge: 9

Press any key to continue

*Програма № 3.*

*// Використання адреси, яка зберігається в покажчику*

#include <iostream>

using namespace std;

int main()

{

 unsigned short int myAge = 5, yourAge = 10;

 unsigned short int \*pAge= &myAge; //

 cout<<"myAge: \t"<<myAge<<"\t yourAge:\t"<<yourAge<<"\n\n";

 cout<<"&myAge: \t"<<&myAge<<"\t&yourAge:\t"<<&yourAge<<"\n\n";

 cout<<"pAge:\t"<<pAge<<"\n\n";

 cout<<"\*pAge:\t"<<\*pAge<<"\n\n";

 pAge=&yourAge;

 cout<<"myAge: \t"<<myAge<<"\t yourAge:\t"<<yourAge<<"\n\n";

 cout<<"&myAge: \t"<<&myAge<<"\t&yourAge:\t"<<&yourAge<<"\n\n";

 cout<<"pAge:\t"<<pAge<<"\n\n";

 cout<<"\*pAge:\t"<<\*pAge<<"\n\n";

 cout<<"&pAge:\t"<<&pAge<<"\n\n";

 cin.get();

 return 0;

}

*Тестування програми*

myAge: 5 yourAge: 10

&myAge: 0012FF7C &yourAge: 0012FF78

pAge: 0012FF7C

\*pAge: 5

myAge: 5 yourAge: 10

&myAge: 0012FF7C &yourAge: 0012FF78

pAge: 0012FF78

\*pAge: 10

&pAge: 0012FF74

Press any key to continue

***II Для чого потрібні покажчики.***

Вище ми детально розглянули процедуру присвоєння покажчикові адреси іншої змінної. Однак на практиці таке використання покажчиків зустрічається досить рідко. До того ж, навіщо задіяти ще і покажчик, якщо значення вже зберігається в іншій змінній? Розглянуті вище приклади приведені тільки для демонстрації механізму роботи покажчиків. Тепер, після опису синтаксису, використовуваного в С++ для роботи з покажчиками, можна переходити до більш професійного їх застосування. Найбільш часто покажчики застосовуються в наступних випадках:

• для розміщення даних у вільних областях пам'яті і доступу до них;

• для доступу до змінних і функцій класів;

• для передачі параметрів у функції за посиланням.

***2.1 Пам'ять стекова і динамічно розподільна.***

Пам’ять умовно поділяється на 5 областей:

• область глобальних змінних;

• вільна або пам'ять динамічно розподільна;

• регістрова пам'ять (регістри);

• сегменти пам’яті;

• стекова пам'ять.

Локальні змінні і параметри функцій розміщаються в **стековій** пам'яті.

Програмний код зберігається в **сегментах**.

Глобальні змінні — в **області глобальних змінних**.

**Регістрова пам'ять** призначена для збереження внутрішніх службових даних програми, таких як адреса вершини стека або адреса команди.

Інша частина пам'яті складає так звану *вільну пам'ять* — область пам'яті, яка розподіляється **динамічно** між різними об'єктами.

Особливістю локальних змінних є те, що після виходу з функції, у якій вони були оголошені, пам'ять, виділена для їхнього збереження, звільняється, а значення змінних знищуються.

Глобальні змінні, на відміну від локальних, зберігають своє значення після виходу із функції і доступні з будь-якої точки програмного коду.

Динамічна пам’ять або вільна – це область пам’яті, розділена на велику кількість пронумерованих комірок, в яких прописується інформація.

Для виділення пам’яті в області динамічного розподілу (вільна пам’ять) використовується ключове слово **new**. Після new слід вказати тип об’єкта, який буде розміщуватись у пам’яті. Це необхідно для визначення розміру області пам’яті, яка потрібна для збереження об’єкта.

Наприклад, new unsigned short int, написавши такий рядок ми виділили 2 байти пам’яті.

Як результат оператор new повертає адресу виділеного фрагменту пам’яті. Цей адрес повинен присвоюватись покажчику.

Наприклад, для виділення пам’яті в області динамічного розподілу змінній типу unsigned short int слід записати:

unsigned short int \* pPointer;

pPointer = new unsigned short int;

Або записати більш компактніше:

**unsigned short int \* pPointer = new unsigned short int;**

В кожному випадку покажчик pPointer буде вказувати на комірку пам’яті в області динамічної пам’яті, яка містить значення типу unsigned short int. Тепер pPointer можна використовувати як будь-який інший покажчик на змінну цього типу.

Щоб занести у виділену область пам’яті яке-небудь значення, потрібно написати такий рядок:

\* pPointer = 72;

Це означає записати число 72 у пам’ять за адресою, яка зберігається в покажчику pPointer.

Коли пам’ять, виділена під змінну, більше не потрібна, її слід звільнити. Робиться це за допомогою оператора delete, після якого записується ім’я покажчика. Оператор delete звільняє область пам’яті, визначену покажчиком.

Необхідно пам’ятати, що покажчик, на відміну від області пам’яті, на яку він вказує, являється локальною змінною. Через те після виходу із функції, в якій він був оголошений, цей покажчик стане недоступним. А область пам’яті, виділена оператором new при цьому не звільняється. В результаті частина пам’яті стає недоступною. Програмісти називають таку ситуацію „утечка памяти”.

Така назва повністю відповідає дійсності, оскільки до завершення роботи програми цю пам’ять використовувати не можна, вона ніби витікає із вашого комп’ютера.

Щоб звільнити виділену пам’ять, використовується ключове слово delete.

Наприклад,

delete pPointer;

При цьому відбувається не знищення покажчика, а звільнення області пам’яті за адресою, яка записана в ньому.

При звільненні виділеної пам’яті із самим покажчиком нічого не відбувається і йому можна присвоїти іншу адресу.

Наведемо приклад програми, яка демонструє як виділити область пам’яті, використати її, а потім звільнити.

*Програма № 4.*

*// Використання вільної пам’яті*

#include <iostream>

using namespace std;

int main()

{

 int local=5;

 int \*pLocal=&local;

 int \*pHeap = new int;

 \*pHeap=7;

 cout<<"Local variable: "<<local<<"\n";

 cout<<"pLocal: "<<pLocal<<"\n";

 cout<<"\*pLocal: "<<\*pLocal<<"\n";

 cout<<"\*pHeap: "<<\*pHeap<<"\n";

 delete pHeap;

 pHeap=new int;

 \*pHeap=9;

 cout<<"\*pHeap: "<<\*pHeap<<"\n";

 delete pHeap; cin.get();

 return 0;

}

*Тестування програми*

Local variable: 5

pLocal: 0012FF7C

\*pLocal: 5

\*pHeap: 7

\*pHeap: 9

Press any key to continue

*Програма № 5.*

*// Використання оператора new*

#include <iostream>

using namespace std;

int main()

{

 int \*pt=new int;

 \*pt = 568912;

 cout<<"value \*pt = "<<\*pt<<" :location pt= "<<pt<<"\n";

 double \*pd=new double;

 \*pd=10000001.0;

 cout<<"value \*pd= "<<\*pd<<" : location pd = "<<pd<<"\n\n\n";

 cout<<"size of pt = "<<sizeof pt<<": size of\*pt= "<<sizeof\*pt<<"\n";

 cout<<"size of pd ="<<sizeof pd<<": size of \*pd = "<<sizeof\*pd<<"\n";

 delete pt;

 delete pd;

 return 0;

}

*Тестування програми*

value \*pt = 568912 :location pt= 00481DF0

value \*pd= 1e+007 : location pd = 00481B70

size of pt = 4: size of\*pt= 4

size of pd =4: size of \*pd = 8

Press any key to continue

*Аналіз програмного коду*

Програма використовує оператор new для виділення пам’яті об’єктам даних типу int і double. Це відбувається при виконанні програми. Покажчики pd, pt показують на ці два об’єкти даних. Без них неможливо звернутися до виділених ділянок пам’яті. Вирази \*pd, \*pt можна використовувати як змінні.

Розмірність покажчика на тип int співпадає з розмірністю покажчика на тип double: обидва являються просто адресами. Адрес сам по собі визначає тільки початок області пам’яті, в якій зберігається об’єкт, а не його тип або розмір.

Компілятор знає, що значення \*pd відноситься до типу double з розмірністю 8 байт. Значення \*pt відноситься до типу int розмірністю 4 байти.

***2.2 Використання оператора new для створення динамічних масивів.***

Якщо програміст оперує лише одним значенням, простіше оголосити просту змінну, ніж використовувати оператор new і покажчик для управлінням одним невеликим об’єктом даних. Найбільш частіше використовують оператор new для роботи з великими порціями даних. Наприклад з масивами, рядками, структурами.

Відмітимо, що розрізняють масиви *статичні* і *динамічні*. Для статичного масиву відбувається виділення пам’яті в процесі компіляції. Незалежно від того, чи буде програма використовувати створений масив чи ні, пам’ять масив буде займати у будь-якому випадку. Статичний масив оголошують так, наприклад, int Array[12];.

Оператор new дозволяє або створювати масив під час виконання програми, або не створювати, якщо такої необхідності не виникло. Розмір масиву можна вибирати під час виконання програми. Створення масиву під час виконання програми називається динамічним зв’язуванням. А сам масив називають *динамічним*.

Як створити динамічний масив? Нехай необхідно створити масив з десяти змінних типу int:

// оголосимо покажчик на тип int і присвоємо адресу пам’яті (вільної)

int \* pSome = new int [10];

Оператор new повертає адресу першого елементу виділеного блоку. Це значення присвоюється покажчику pSome. Щоб звільнити пам’ять, виділену для масиву потрібно записати:

delete [] pSome;

Квадратні дужки вказують, що потрібно звільнити пам’ять, відведену для цілого масиву, а не для елементу, на який вказує покажчик.

Як використовується масив після його створення?

int \* pSome = new int [10];

Створюється покажчик pSome, який вказує на перший елемент групи з десяти змінних типу int. Уявіть його у вигляді указки, яка направлена на цей елемент.

Припустимо, значення типу int займає чотири байти. Значить, пере двинувши указку на чотири байти у відповідному напрямку, можна вказати на другий елемент. Таким чином, оператор new надає всю інформацію, яка потрібна для ідентифікації кожного елемента групи.

Розглянемо цю проблему з практичної точки зору. Як відбувається звернення до будь-якого з цих елементів? З першим елементом дуже все просто. Оскільки покажчик pSome вказує на перший елемент масиву, то вираз \* pSome являється значенням першого елементу. Залишається ще 9 елементів, до яких потрібно отримати доступ.

Для цього потрібно просто застосувати покажчик так, ніби він являється іменем масиву.

Іншими словами, для першого елементу можна використати позначення pSome[0] замість \*pSome, для другого - pSome[1] і т.д.

Таким чином, можна дуже просто застосовувати покажчик для звернення до динамічного масиву.

Річ в тім, що мова С++ опрацьовує масиви внутрішнім способом, у будь-якому випадку використовуючи покажчики.

Масиви і покажчики тісно зв’язані – в цьому трудність і в цьому сила і перевага мови С++.

*Програма № 6.*

*//Розглянемо приклад створення динамічного масиву*

#include <iostream>

using namespace std;

int main()

{

 //виділяється область вільної пам’яті для трьох елементів типу double

 double \*p3=new double [3];

 // p3 трактується як ім’я масиву

 p3[0]=0.2;

 p3[1]=0.5;

 p3[2]=0.8;

 //вивести значення адреси

 cout<<"Pointer p3 is "<<p3<<"\n\n";

 cout<<"\*p3 is "<<\*p3<<"\n\n"; // розіменувати покажчик, щоб вивести значення першого елементу динамічного масиву

 cout<<"p3[0] is "<<p3[0]<<"\n\n";

// Ім’я покажчика використовується як ім’я масиву

 cout<<"p3[1] is "<<p3[1]<<"\n";

 cout<<"p3[2] is "<<p3[2]<<"\n";

 // збільшення значення покажчика на одиницю

 p3=p3+1;

// Ми можемо доступитися лише до нульового і першого елементу

// Їх значення змінились дещо

 cout<<"Now p3[0] is "<<p3[0]<<"and";

 cout<<"p3[1] is "<<p3[1]<<"\n\n\n";

// Зменшимо значення покажчика на одиницю

 p3=p3-1;

// Вивільнимо пам’ять

 delete []p3;

 return 0;

}

*Тестування програми*

Pointer p3 is 00481DD0

\*p3 is 0.2

p3[0] is 0.2

p3[1] is 0.5

p3[2] is 0.8

Now p3[0] is 0.5 and p3[1] is 0.8

Press any key to continue

Мова С++ опрацьовує масиви внутрішнім способом, застосовуючи покажчики. Подібність між масивами і покажчиками – перевага мови С++.

Якщо ви одиницю додасте до покажчика, то ви збільшите значення покажчика на число байтів, яке відповідає розмірності типу, на який він вказує. Продемонструємо цей факт у наступному прикладі.

*Програма № 7.*

*//У мові С++ ім’я масиву інтерпретується як адрес його першого елементу*

#include <iostream>

using namespace std;

int main()

{

 double w[3]={11111.1, 22222.2, 333333.3};

 short s[3]={1,2,3};

 //ім’я масиву є адреса

 double \*p\_w=w;

 short \*p\_s=&s[0];

 // визначаємо адресу для нульового елементу

 cout<<"p\_w = "<<p\_w<<", \*p\_w = "<<\*p\_w<<"\n\n";

 //Якщо додати 1 до покажчика

 p\_w=p\_w+1;

 cout<<"add 1 to the p\_w pointer: \n\n";

 cout<<"p\_w = "<<p\_w<<", \*p\_w= "<<\*p\_w<<"\n\n";

 cout<<"p\_s= "<<p\_s<<", \*p\_s = "<<\*p\_s<<"\n\n";

 p\_s=p\_s+1;

 cout<<"add 1 to the p\_s pointer :\n\n";

 cout<<"p\_s="<<p\_s<<", \*p\_s="<<\*p\_s<<"\n\n";

 cout<<"access two elements with array notation\n\n";

 cout<<"s[0]="<<s[0]<<" "<<"s[1]="<<s[1]<<"\n\n";

 cout<<"access two elements with pointer notation\n\n";

 cout<<"\*s= "<<\*s<<" "<<"\*(s+1) = "<<\*(s+1)<<"\n\n";

 cout<<sizeof w<<" = size of w array\n\n";

 cout<<sizeof p\_w<<" = size of p\_w pointer \n\n";

 return 0;

}

*Тестування програми*

p\_w = 0012FF68, \*p\_w = 11111.1

add 1 to the p\_w pointer:

p\_w = 0012FF70, \*p\_w= 22222.2

p\_s= 0012FF60, \*p\_s = 1

add 1 to the p\_s pointer :

p\_s=0012FF62, \*p\_s=2

access two elements with array notation

s[0]=1 s[1]=2

access two elements with pointer notation

\*s= 1 \*(s+1) = 2

24 = size of w array

4 = size of p\_w pointer

Press any key to continue

У мові С++ ім’я масиву елементів типу char, покажчик на елемент char і рядкова константа в лапках інтерпретується як адрес першого символу рядка.

Продемонструємо це на прикладі програми:

*Програма № 8.*

*//У мові С++ ім’я масиву елементів типу char, покажчик на елемент char*

*// і рядкова константа в лапках інтерпретується як адрес першого символу*

*// рядка.*

#include <iostream>

using namespace std;

int main()

{

 char animal[20]= "bear";

 const char \*bird= "wren";

 char \*ps;

 cout<<animal<<" and ";

 cout<<bird<<"\n";

 //cout<<ps<<"n"; // помилка – спроба вивести значення покажчика, якому нічого не присвоєно

 cout<<"Enter a kind of animal: "; // Введіть вид тварин

 cin>>animal;

 //cout<<ps<<"!s\n";

 cout<<"Before using strcpy():\n";

 cout<<animal<< " at "<<(int \*)animal<<endl;

 //cout<<ps<<" at"<<(int\*)ps<<endl;

 ps=new char[strlen(animal)+1];

 // виділити для рядка область пам’яті потрібного розміру

 strcpy(ps,animal);

 // копіюємо рядок із масиву animal в щойно виділену область пам’яті

 cout<<"After using strcpy():\n";

 cout<<animal<<" at "<<(int \*)animal<<endl;

 cout<<ps<<" at "<<(int\*)ps<<endl;

 delete []ps;

 return 0;

}

*Тестування програми*

bear and wren

Enter a kind of animal: crocodile

Before using strcpy():

crocodile at 0012FF6C

After using strcpy():

crocodile at 0012FF6C

crocodile at 00481C80

Press any key to continue

*Аналіз програмного коду*

У програмі створюється один масив типу char ( animal ) і два покажчики на тип char ( bird і ps ). Масиву animal присвоюється рядок ( char animal[20]="bear"; ).

const char \*bird="wren";

Покажчику присвоюється рядок, а точніше присвоюється адреса. Адже, рядкова константа в лапках "wren" являє собою адресу першого символу рядка.

Рядкові літерали являються константами, через те в коді при їх оголошенні використовують ключове слово const.

cout<<animal<<" and ";

cout<<bird<<"\n";

Із програми видно, що в операторі cout можна використовувати ім’я масиву animal і покажчик bird рівноцінно. Об’єкт cout відобразить два рядки "bear" і "wren".

Масив animal можна використовувати для введення при умові, що введений рядок не перебільшує 20 символів.

cout<<"Enter a kind of animal: "; // Введіть вид тварин

cin>>animal;

Проте, було б не коректно використовувати для введення покажчик bird. Оскільки покажчик bird оголошено із специфікатором const, компілятор попереджає будь які спроби змінити зміст тієї адреси, на яку вказує bird.

cout<<animal<<" at "<<(int \*)animal<<endl;

Зазвичай якщо в операторі cout використовується покажчик, виводиться адреса. Але, якщо тип покажчика - char \*, оператор cout відображає рядок, на який вказує покажчик. Щоб побачити адресу рядка, слід зробити зведення типу покажчика до покажчика іншого типу, такого як int \*.

***2.3 Використання оператора new для створення динамічних структур.***

Оператор new дозволяє створювати динамічні структури. Це означає, що пам’ять виділяється під час виконання програми, а не під час компіляції.

Використання оператора new при роботі з структурами проходить два етапи: створення структури і доступ до її елементів.

Наведу приклад програми:

*Програма № 9*

*//Використання оператора new для створення динамічних структур.*

#include <iostream>

using namespace std;

struct inflatable // шаблон структури

{

 char name[20];//елементи структури

 float volume;

 double price;

}; // крапка з комою завершує оголошення структури

int main()

{

 inflatable \*ps=new inflatable;

 //виділення області пам’яті для структури

 //покажчику ps присвоюється адреса частини вільної пам’яті,

 //достатньої для збереження структури типу inflatable

 //”Введіть назву товару”

 cout<<"Enter name of inflatable item:";

 cin.get(ps->name ,20);

 //оператор належності у вигляді -> дає змогу доступитись до

 //елементів структури

 //Отже, покажчик ps вказує на структуру типу inflatable.

 //Тоді, ps->price вказує на елемент price вказаної структури inflatable.

 //

 cout<<"Enter volume in cubic feet: ";

 cin>>ps->volume ;

 cout<<"Enter price: $";

 cin>>ps->price ;

 cout<<"Name: "<<ps->name <<"\n";

 cout<<"Volume : "<<ps->volume <<"cubic feet\n";

 cout<<"Price:"<<ps->price <<"\n";

 return 0;

}

*Тестування програми*

Enter name of inflatable item:Table

Enter volume in cubic feet: 15

Enter price: $56.23

Name: Table

Volume : 15cubic feet

Price:56.23

Press any key to continue

*Аналіз програмного коду*

Щоб доступитися до елементів динамічної структури не можна застосовувати оператор належності у вигляді крапки, оскільки у структури немає імені. Для цього використовується інший оператор належності у вигляді стрілки. Наприклад, ps->name.

Але, якщо ps – покажчик на структуру, то \* ps – значення, на яке вказує покажчик. Тоді (\* ps). price – елемент структури. Отже, ми отримали другий спосіб доступу до елементів динамічної структури.

Програма може бути такою:

#include <iostream>

using namespace std;

struct inflatable

{

 char name[20];

 float volume;

 double price;

};

int main()

{

 inflatable \*ps=new inflatable;

 cout<<"Enter name of inflatable item:";

 cin.get((\*ps).name ,20);

 cout<<"Enter volume in cubic feet: ";

 cin>>(\*ps).volume ;

 cout<<"Enter price: $";

 cin>>(\*ps).price ;

 cout<<"Name: "<<(\*ps).name <<"\n";

 cout<<"Volume : "<<(\*ps).volume <<"cubic feet\n";

 cout<<"Price:"<<(\*ps).price <<"\n";

 return 0;

}

Взагалі у мові С++ є три складених типи – це масиви, структури, покажчики. В масиві може зберігатись декілька значень одного типу і того ж типу. Для звернення до окремих елементів масиву застосовують індекс.

Рядок – це масив символьного типу. Рядок являє собою послідовність символів, які завершуються нульовим символом. Він може бути представлений рядковою константою в лапках; в цьому випадку нульовий символ неявно мається на увазі. Рядок можна зберігати в масиві символьного типу, а також представити у вигляді покажчика на дані типу char. Функція strlen() повертає довжину рядка без врахування нульового символу. Функція strcpy() копіює рядок із однієї пам’яті в іншу. При використанні цих функцій потрібно включати файл заголовку cstring.

Структура може містити декілька значень різних типів в одному об’єкті даних. Для звернення до окремих елементів структури застосовується оператор належності (.). Спочатку структуру слід оголосити, тобто створити шаблон структури. Шаблон структури визначає, які елементи вона може містити. Ім’я або дескриптор шаблону структури стає ідентифікатором нового типу даних. Далі можна оголошувати структурні змінні цього типу.

Покажчики – це змінні, в яких зберігається адреса. Ми говоримо, що покажчик вказує на адресу, яку він містить. При оголошенні покажчика необхідно вказати, на об’єкт якого типу він вказує. Результатом застосування операції розіменування (\*) до покажчика являється значення, яке зберігається за даною адресою.

Оператор new дозволяє зарезервувати пам’ять для об’єкту даних під час виконання програми. Цей оператор повертає адресу отриманої області пам’яті і присвоює покажчику. Єдиний спосіб доступитися до цієї області пам’яті – це використати покажчик. Якщо об’єкт даних проста змінна, можна використати операцію розіменування (\*) для того щоб отримати значення, яке зберігається за цією адресою. Якщо об’єкт даних являє собою масив, для доступу до його елементів можна використати покажчик так, ніби він являється іменем масиву. Якщо об’єкт даних являє собою структуру, то для доступу до його елементів можна використовувати операцію розіменування покажчика (->).

Покажчики і масиви тісно пов’язані між собою. Якщо, наприклад, ar – це ім’я масиву, то вираз ar [i] інтерпретується як \*(ar + i ), а масив – як адрес першого елементу масиву. Таким чином, ім’я масиву відіграє ту ж саму роль, що і покажчик. В свою чергу, можна використовувати ім’я покажчика і синтаксис позначення масиву для звернення до елементів масиву, який створений за допомогою оператора new.

Оператори new і delete дають можливість явно управляти виділенням пам’яті для об’єктів даних і вивільненням її.

Використані джерела інформації

1. Джесс Либерти “С++ за 21 день”, М., 2011.
2. Стивен Прата “Язык программирования С++. Лекции и упражнения”, DiaSoft, М., 2013
3. Митчелл “Толковый словарь компьютерных технологий”
4. Вирт Н. Алгоритмы и структуры данных / Пер. с англ. – М.: Мир, 1989. – 360 с., ил.
5. Гринзоу Лу. Философия программирования для Windows 95/NT / Пер. с англ.- СПб.: Символ-Плюс, 1997. – 640 с., ил.
6. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил.
7. Зелковиц М., Шоу А., Геннон Дж. Принципы разработки программного обеспечения / Пер. с англ. –М.: Мир, 1982. – 386 с., ил.
8. Ковалюк Т. В. Основи програмування. – К.: Видавнича група BHV,2013/ - 384 c.: іл.