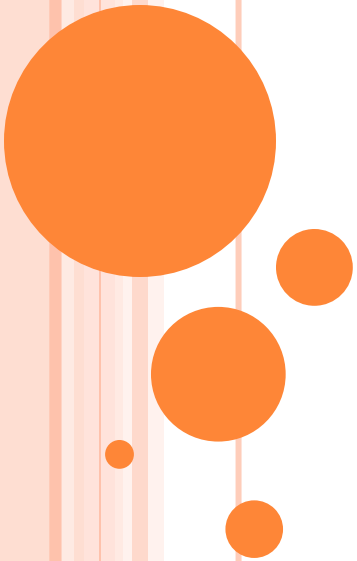


Лекція 21

Динамічні структури даних

Динамічні структури даних

- ***Динамічні структури даних*** - це структури даних, пам'ять під які виділяється і звільняється в міру необхідності.
- Динамічні структури даних в процесі існування в пам'яті *можуть змінювати не тільки число складових їх елементів, а й характер зв'язків між елементами*.
- При цьому не враховується зміна вмісту самих елементів даних. Така особливість динамічних структур, як зміна їх розміру і характеру відносин між елементами, призводить до того, що на етапі створення машинного коду програма-компілятор не може виділити для всієї структури в цілому ділянку пам'яті фіксованого розміру, а також не може зіставити з окремими компонентами структури конкретні адреси.

Динамічні структури даних

- ***Динамічна структура даних характеризується тим що***:
  1. вона не має імені;
  2. їй виділяється пам'ять в процесі виконання програми;
  3. кількість елементів структури може не фіксуватися;
  4. розмірність структури може змінюватися в процесі виконання програми;
  5. в процесі виконання програми може змінюватися характер взаємозв'язку між елементами структури.

Динамічні структури даних

- ***К***ожній динамічної структурі даних **зіставляється статична змінна типу вказівник** (її значення - адреса цього об'єкта), за допомогою якої здійснюється доступ до динамічної структурі.

- ***Необхідність в динамічних структурах даних зазвичай виникає в наступних випадках.***
  1. Використовуються змінні, що мають досить великий розмір (наприклад, масиви великої розмірності), необхідні в одних частинах програми і абсолютно не потрібні в інших.
  2. В процесі роботи програми потрібен масив, список або інша структура, розмір якої змінюється в широких межах і важко передбачуваний.
  3. Коли розмір даних, що обробляються в програмі, перевищує обсяг сегмента даних.

Динамічні структури даних

- Динамічні структури, за визначенням, ***характеризуються відсутністю фізичної суміжності елементів структури в пам'яті, непостійністю і непередбачуваністю розміру*** (числа елементів) структури в процесі її обробки.
- Оскільки елементи динамічної структури розташовуються по непередбачуваним адресами пам'яті, адреса елемента такої структури не може бути обчислений з адреси початкового або попереднього елемента.
- Для встановлення зв'язку між елементами динамічної структури використовуються вказівники, через які встановлюються явні зв'язки між елементами. Таке уявлення даних в пам'яті називається *зв'язковим.*

Динамічні структури даних

- Переваги зв'язкового представлення даних - в можливості забезпечення значною мінливості структур:

1. розмір структури обмежується тільки доступним об'ємом машинної пам'яті;
2. при зміні логічної послідовності елементів структури потрібно не переміщення даних в пам'яті, а тільки корекція вказівників;
3. велика гнучкість структури.

Динамічні структури даних

- Недоліки:

1. на поля, що містять вказівники для зв'язування елементів один з одним, збільшують споживання пам'ять;
2. доступ до елементів зв'язної структури може бути менш ефективним за часом.

Динамічні структури даних

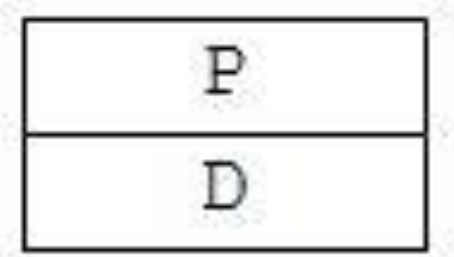
- **Порядок роботи з динамічними структурами даних наступний**:
- створити (відвести місце в динамічній пам'яті);
- працювати за допомогою вказівника;
- видалити (звільнити зайняте структурою місце).

Класифікація динамічних структур даних

- ***До таких структур відносять:***
  - односпрямовані (одинзв'язні) списки;
  - двонаправлені (двусвязного) списки;
  - циклічні списки;
  - стек;
  - дек;
  - черга;
  - бінарне дерева.

Оголошення динамічних структур даних

- Кожна компонента будь-динамічної структури є запис, що містить, принаймні, два поля: одне поле типу вказівник, а друге - для розміщення даних.
- Для найкращого представлення зобразимо окрему компоненту у вигляді:



- де поле P - вказівник; поле D - дані.

Оголошення динамічних структур даних

- Елемент динамічної структури складається з двох полів:
  1. інформаційного поля (поля даних), в якому містяться ті дані, заради яких і створюється структура; в загальному випадку інформаційне поле саме є інтегрованою структурою - вектором, масивом, інший динамічною структурою і т.п .;
  2. адресного поля (поля зв'язок), в якому містяться один або кілька вказівників, що зв'язує даний елемент з іншими елементами структури.

Оголошення динамічних структур даних

- Оголошення виглядає наступним чином:

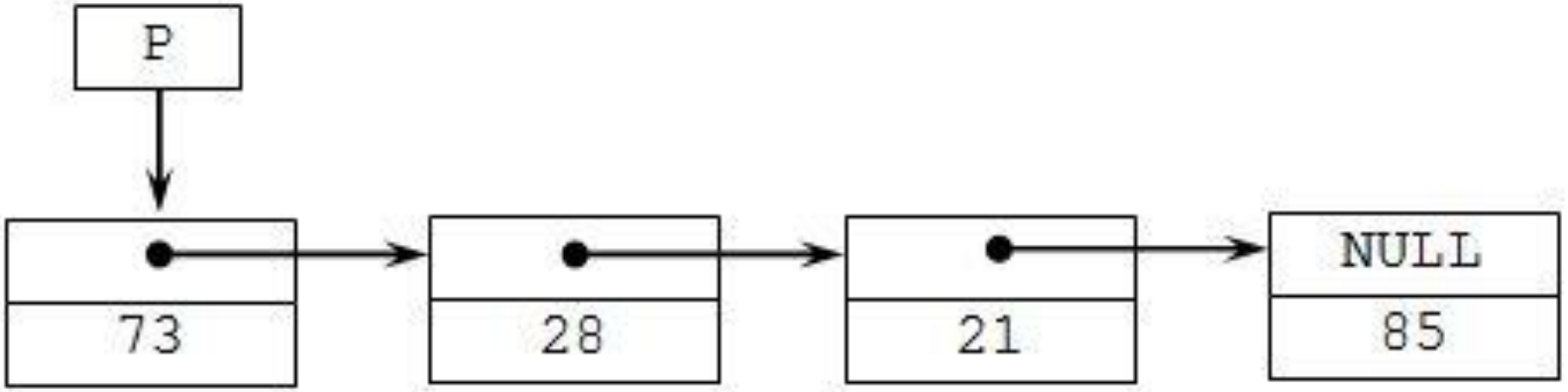
```
struct імя_тіпа
 {
        інформаційне поле;
        адресне поле;
  };
```

- наприклад:

```
struct TNode
      {
         int Data;        // інформаційне поле
         TNode * Next;      // адресне поле
       };
```

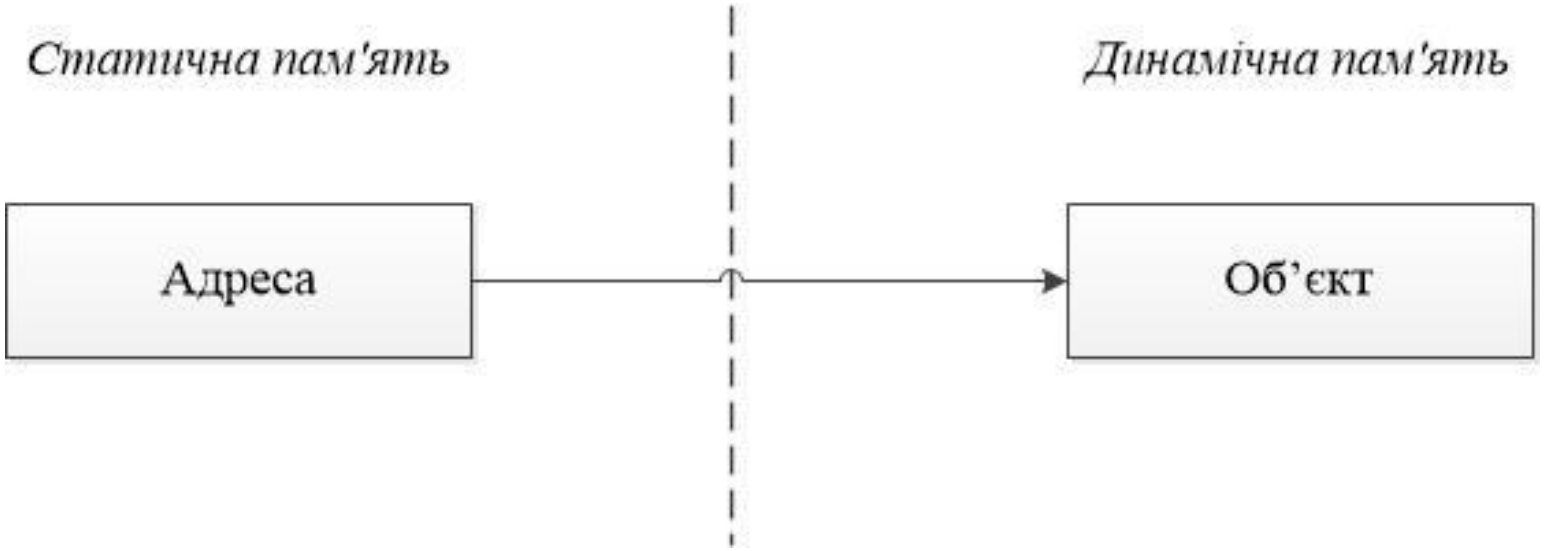
Інформаційних і адресних полів може бути як одне, так і декілька.

Оголошення динамічних структур даних



Доступ до даних в динамічних структурах

Вказівник містить адресу певного об'єкта в динамічної пам'яті. Адреса формується з двох слів: адреса сегмента і зміщення. Сам вказівник є статичним об'єктом і розташований в сегменті даних



Оголошення динамічних структур даних

- Доступ до даних в динамічних структурах здійснюється за допомогою операції "стрілка" (->), яку **називають операцією непрямого вибору елемента структурного об'єкта, що адресується вказівником**.

- Формат застосування даної операції наступний:
- Вказівник_На_Структуру-> Імя_Елемента

Оголошення динамічних структур даних

- Операції "стрілка" (->) двомісна. Застосовується для доступу до елементу, що задається правим операндом, **тієї структури, яку адресує лівий операнд**. Як лівого операнда повинен бути вказівник на структуру, а в якості правого - **ім'я елемента цієї структури**.
- Наприклад:
- p-> Data;
- p-> Next;
- **Необхідно пам'ятати,** що робота з динамічними даними уповільнює виконання програми, оскільки *доступ до величини відбувається в два етапи*: спочатку шукається **вказівник**, потім по ньому - **величина**.

Робота з пам'яттю при використанні динамічних структур

```
struct Node
{  char * Name;
   int Value;
   Node * Next
 };
Node * PNode;       // оголошується вказівник

PNode = new Node;    // виділяється пам'ять

PNode-> Name = "STO";  // присвоюються значення
PNode-> Value = 28;
PNode-> Next = NULL;

delete PNode;        // звільнення пам'яті
```

Алгоритм організації динамічних структур даних

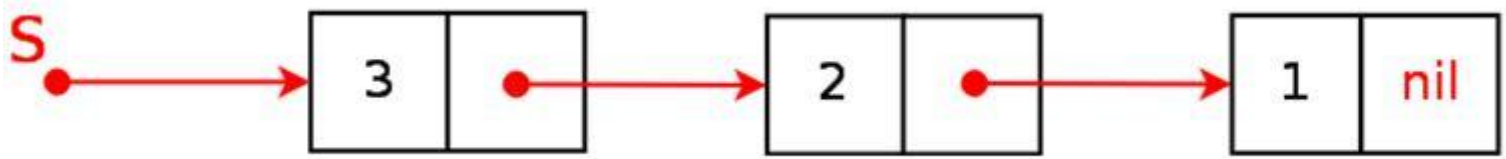
- Для динамічних типів даних **не оголошуються змінні, інакше пам'ять б виділялася під змінні**.

Алгоритм організації динамічних структур даних

- Ідея полягає в наступному:

1. У програмі визначаємо структурний тип даних з "родзинкою" і створюємо змінну вказівник на нього. В результаті при запуску програми пам'ять виділяється тільки під вказівник.
2. У процесі виконання програми, в разі виникнення необхідності в створенні структури, за допомогою спеціальної функції виділяємо пам'ять під зберігання даних (полів структури).
3. Надаємо вказівником адресу, за якою розташована щойно створена структура.
4. Коли надходить команда на створення такої структури, знову за допомогою функції виділяється пам'ять, а вказівником присвоюється адреса цієї нової структури.
5. "Родзинка" певного раніше структурного типу даних полягає в тому, що одним з його полів є вказівник на структуру цього ж типу.
6. У це поле-вказівник записується адреса на структуру, яка була створена перед даною структурою.

Алгоритм організації динамічних структур даних



Функція malloc

- Функція **malloc ()** повертає адресу на перший байт області пам'яті розміром size байт, яка була виділена з купи.
- Якщо пам'яті недостатньо, щоб задовольнити запит, функція malloc () повертає **нульовий вказівник**.
- *Дуже важливо завжди перевіряти повертається значення на його рівність NULL*, перш ніж намагатися використовувати цей вказівник.
- Формат звернення до цієї процедури:
- *ідентифікатор* =
  - = *(тип_ідентіфікатора *) malloc (sizeof (тип_ідентіфікатора))*
- Вважається, що після виконання цього оператора створена динамічна величина, ім'я якої має такий вигляд:
- ***<Ім'я динамічної величини> = * <вказівник>***

Функція malloc

- Нехай в програмі, в якій є наведене вище, присутні наступні оператори:
- P1 = (int *) malloc (sizeof (int));
- P2 = (char *) malloc (sizeof (char));
- Після їх виконання у динамічній пам'яті виявляється виділеним місце під три величини (два скалярні і один масив), які мають ідентифікатори:
- * P1, * P2
- Наприклад,
- позначення * P1 можна розшифрувати так: Динамічна змінна, на яку посилається вказівник P1.

Функція malloc

- Подальша робота з динамічними змінними відбувається точно так само, **як зі статичними змінними відповідних типів**. Ї
- м можна присвоювати значення, їх можна використовувати в якості операндів у виразах, параметрів підпрограм і пр.
- Наприклад, якщо змінної * P1 потрібно присвоїти число 25, змінної * P2 привласнити значення символу "A", то це робиться так:
- * P1 = 25;
- * P2 = 'A';
- Крім процедури malloc значення вказівника може визначатися оператором присвоювання:
- *<Вказівник> = <Посилання на вираз>;*

Функція malloc

Як посиланя на вираз можна використовувати

1. вказівник;
2. кількість посилань функцію (тобто функцію, значенням якої є вказівник);
3. константу NULL.

- **NULL** - це зарезервований константа, що позначає порожню посилання, тобто посилання, яка ні на що не вказує. При присвоєнні базового типу вказівника та посилання на вирази повинні бути однакові.
- Константу NULL можна привласнювати вказівником з **будь-яким базовим типом.**
- Нехай в програмі описані наступні вказівники:
- int * D, * P;
- D = P; K = NULL;

Функція malloc

- **D = (int *) malloc (sizeof (int));**
- **P = (int *) malloc (sizeof (int));**
- {Виділено місце в динамічній пам'яті під дві цілі змінні.}
- {Вказівники отримали відповідні значення}
- *** D = 3; * P = 5;**
- {Динамічним змінним присвоєно значення}
- **P = D;**
- {Вказівники P і D стали посилатися на одну і ту ж величину, рівну 3}
- **cout << * P << * D;** {Двічі друкується число 3}

Функція calloc

- Функція calloc () **повертає вказівник на виділену пам'ять**. Розмір виділеної пам'яті дорівнює величині num * size, де size задається в байтах. Це означає, що функція calloc () **виділяє достатньо пам'яті для масиву з num об'єктів кожен розміром size байт.**
- Функція calloc () **повертає вказівник на перший байт виділеної області**. Якщо пам'яті недостатньо для задоволення запиту, то повертається нульовий вказівник. Завжди важливо перевірити, чи значення, що повертається, на його рівність **NULL, перш ніж використовувати цей вказівник.**
-
- void *calloc(size_t num, size_t size)

Функція calloc

- Наступна функція повертає вказівник на динамічно виділений масив для 100 чисел типу float:

```
#include <stdlib.h>
#include <stdio.h>
float *get_mem(void)
{
float *p;
p = (float *) calloc(100, sizeof(float));
if(!p) {
  printf ("Allocation failure.");
exit (1);
}
return p;
}
```

Функція calloc

- Наступна функція повертає вказівник на динамічно виділений масив для 100 чисел типу float:

```
#include <stdlib.h>
#include <stdio.h>
float *get_mem(void)
{
float *p;
p = (float *) calloc(100, sizeof(float));
if(!p) {
  printf ("Allocation failure.");
exit (1);
}
return p;
}
```

Функція free

- Функція free **звільняє місце в пам'яті**. Блок пам'яті, раніше виділений за допомогою виклику malloc, calloc або realloc звільняється. Тобто звільнена пам'ять може далі використовуватися програмами або ОС.
- Зверніть увагу, що ця функція залишає значення ptr незмінним, отже, він як і раніше вказує на той же блок пам'яті, а не на нульовій вказівник.
- параметри:
- Ptrmem - Вказівник на блок пам'яті, раніше виділений функціями malloc, calloc або realloc, яку необхідно вивільнити. Якщо в якості аргументу передається нульовий вказівник, ніяких дій не відбувається.
- Функція не має значення, що повертається.
-
- void free(void * ptrmem);

Функція free

```
#include <iostream>
#include <cstdlib>

int main ()
{
   int * buffer1 = (int *) malloc (100 * sizeof (int)),
// виділяємо пам'ять під 100 елементів масиву типу int, з попередньою ініціалізацією
        * Buffer2 = (int *) calloc (100, sizeof (int)),
 // виділяємо пам'ять під 100 елементів масиву типу int, без ініціалізації
        * Buffer3 = (int *) realloc (buffer2, 500 * sizeof (int));
// перерозподілити пам'ять в блоці buffer2, новий розмір блоку - 500 елементів

   free (buffer1); // вивільняє блок пам'яті buffer1
   free (buffer3); // вивільняє блок пам'яті buffer2, його нова адреса, після перерозподілу, зберігається в buffer3  return 0;
}
```

Дякую за увагу!