Структури та зв`язні списки

Проф. Куссуль Н.М.

Структури в С++

- **Структура в С ++** - складений (користувацький) тип даних, який може містити дані різних типів (простих та користувацьких).
- Використовуються для логічного та фізичного **об'єднання даних довільних типів**, так само як масиви служать для групування даних одного типу.
- Структура в С++ може містити також функції.

Структури в C++

- **Формат опису**

```
struct <ім'я_структури> {
        елементи структури
   };
```

- **Приклад**

```
struct Student
   {
        int age;
        string name;
   };
```

Ідентифікатор структури в C++ є іменем **типу**

Ініціалізація структур

Створення змінної типу Student

```
Student stud1;
stud1.age = 21;
stud1.name = “Jenny Doe”;
```

Ініціалізація змінної типу Student

```
Student  stud2 = {22, “John Doe”};
```

- Елементи структури називаються **полями**.
- Звертання до полів за оператором **.** (**точка**)

Оголошення змінних типу структури та синонімів імені типу

- В С можна оголошувати змінні структури при описі структури:

```
struct Student // тип даних структура
 {
    string name;
    int age;
} s1, s2;
```

- s1, s2 – **змінні** типу структури
- **typedef** struct Tdata1

```
{    int a;
     data2 *d; }  data1; // синонім імені типу
```

Створення структури через вказівник

- **Вказівник на структуру**

```
Student* stud3;
stud3= new Student; //створення структури
```

- **Звертання до полів структури**

```
stud3->age = 25;
stud3->name = “Johnny Doe”;
```

- **або**

```
(*stud3).age = 25;
(*stud3).name = “Johnny Doe”;
```

Приклади роботи зі структурами

```
struct employee //оголошення структури
 { char name [64];
   long employee_id;
 } worker; //визначення об'єкта типу employee
strcpy(worker.name, “James Bond");// копіювати ім‘я
worker.employee_id = 12345;
void show_employee(employee worker)
   //оголошення функції, що виводить елементи
   структури
{ cout << " Name" << worker.name;
  cout << " id" << worker.employee_id; }
show_employee(worker);//виклик функції
```

Приклади роботи зі структурами

- **void get_employee_id**(employee ***worker**)

{ cout << "Введите номер служащего: ";

cin >> **worker->**employee_id; } /* програма передає змінну worker типу структури у функцію get employee_id за допомогою вказівника */

get_employee_id(**&worker**) ;

Якщо **функція змінює** елемент структури, то у функцію повинна передаватися адреса та використовуватися вказівник на структуру. Для звернення до елемента структури функції використовується формат:

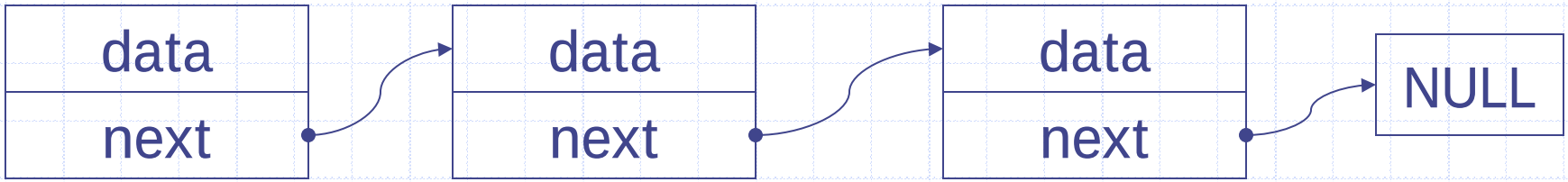
```
value = variable-> member;
variable-> other_member = some_value;
```

Зв’язний список - означення

- **Зв’язний список** (linked list)
  - Це структура даних, в якій об`єкти розташовуються в лінійному порядку
  - На відміну від масиву, у якому цей порядок задається індексами, **порядок в зв’язному списку задається вказівниками на кожен об’єкт**
- Зв’язні списки забезпечують просте та гнучке представлення динамічних множин.

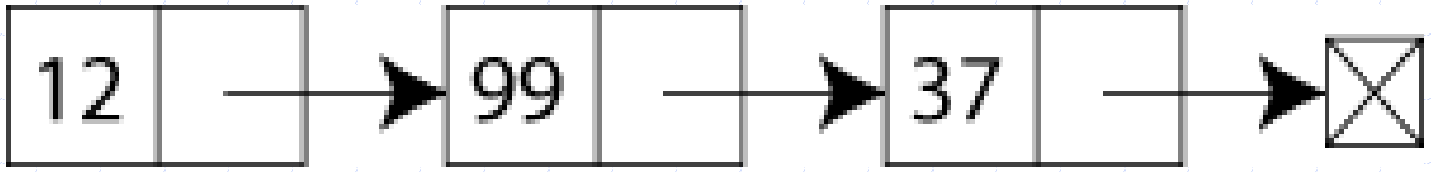
Різновиди зв'язних списків

- **Однозв’язний список (singly linked list)**
  - це такий список, у якого кожен елемент зберігає в собі вказівник на наступний після нього елемент.
- Вказівник **останнього елемента** списку вказує на порожній елемент, тобто рівний **NULL**.



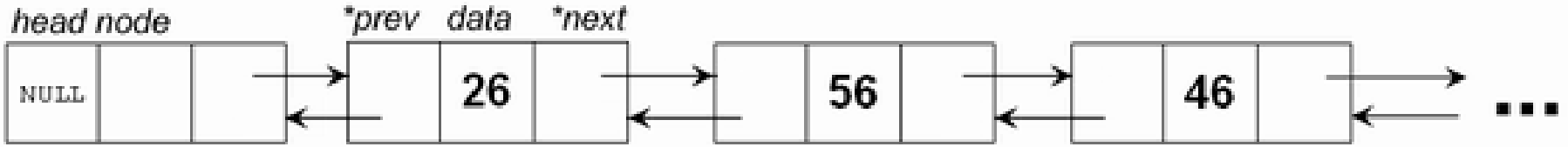
Однобічно зв'язані списки

- В однобічно зв'язаному списку, який є найпростішим різновидом зв'язаних списків, кожний елемент складається з двох полів:
- *data* або даних, та
- вказівника *next* на наступний елемент.
- Якщо вказівник не вказує на інший елемент (інакше: *next = NULL*), то вважається, що даний елемент — останній в списку.



Двобічно зв'язаний список

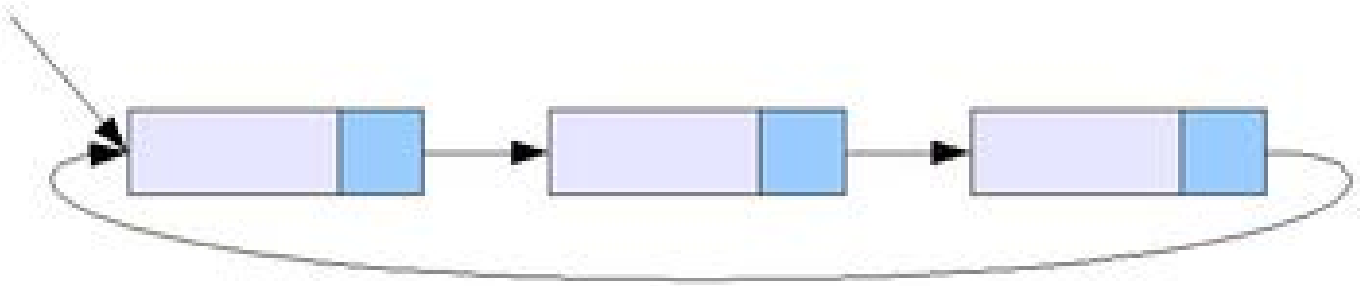
- В двобічно зв'язаному списку елемент складається з трьох полів —
  - **вказівника на попередній** елемент **`prev`**,
  - поля **даних** **`data`** та
  - вказівника **`next`** **на наступний елемент**.
  - Якщо *prev=NULL*, то в елемента немає попередника (тобто він є «головою» списку), якщо *next=NULL*, то в нього немає наступника («хвіст» списка).



Різновиди зв'язних списків

Кільцевий список

- В кільцевому списку перший та останній елемент зв'язані. Тобто, поле *prev* голови списка вказує на хвіст списка, а поле *next* хвоста списка вказує на голову списка.



Приклад – список рядків

```
struct element          //оголошення
{                       //структури
   string str;
   element* next;
};
```

- Однозв'язний список
  - елемент – рядки типу string
  - вказівник на наступний елемент списку

Приклад реалізації списку рядків

```
// FILE module.h
#include <iostream>
#include <string>
#include <conio.h>

using namespace std;

    struct element{ // оголошення типу елемента списка
            string str;
            element* next;
    };

element* EnterList();
int Count(element*);
```

Створення списку рядків та його виведення

```
// FILE project1.cpp with main() function
#include "module.h"

int main()
{
    element *current, *top;
    top = EnterList();
    current = top;
    while (current!=0)
    {
        cout<< current->str << endl;
            current=current->next;
    }

    cout << endl << "Size of List is " << Count(top) << endl;
    getch();
    return 0;
}
```

Створення списку рядків

```
// FILE module.cpp with functions
#include "module.h"

element* EnterList()
{
    element *first, *current;
    string answer;

    cout<< "enter first string : ";
    first=current=new element;
    cin>> current->str;
    cout<< "do you want new
string? (n for exit)";
    cin>> answer;
while(answer != "n")
{
    current->next = new element;
    current = current->next;
    cout <<          "enter string :  ";
    cin>>current->str;
    cout<< "do you want new string?
                              (n for exit)";
                    cin>> answer;
}

  current->next = NULL;
  return first;
}
```

Обчислення кількості елементів списку

```
int Count (element* list)
{
    int result = 0;

    if (list == NULL)       {
            cout << "List is empty";
    }
    while (list!=0)         {
        result++;
            list=list->next;
    }
    return result;
}
```

Однозв'язний список цілих чисел: приклади операцій

- Список зручно використовувати для динамічних структур даних, якщо невідомі розміри та кількість елементів

```
struct  list {
        int value;
        list* next;
}; // список ненульових цілочисельних елементів
```

- Заповнення списку

```
for (int i=0; list->value != 0; list->next = new list)
        {
                list=list->next;
                list->next=NULL;
                cin>>list->value;
        }
```

Пошук максимального елементу списку

```
struct  list {
        int value;
        list* next;
    };


int Maximum (list* lst)
    {
        int max = lst->value;

        for(int i=0; lst != NULL; lst = lst->next)
         {
                if (lst->value > max)
                  max = lst->value;
         }
        return  (max);
    }
```

Однозв'язний список

- У однозв'язному списку відсутній вказівник на попередній елемент. Тому **варто зберігати вказівник на початок списку**
- Функція, яка повертає вказівник на **попередній** елемент

```
list* prev(list* head, list* cur)
{ //head – вказівник на початок списку
  //cur – поточний елемент
        list* lst;
        lst = new list;
        lst = head;
        lst->next = head->next;

        for(int i=0; lst->next != cur; lst=lst->next)
        {
                if (lst->next==NULL) break;
        }
        return (lst);
}
```

Однозв'язний список

- Додавання елементу **після** поточного

```
list* lst;
lst=new list;
lst->next=cur->next;
cur->next=lst;
```

- Додавання елементу **перед** поточним

```
list* lst;
lst=new list;
lst->next=prev(head,cur)->next;
prev(head,cur)->next=lst;
```

Однозв'язний список - пошук елемента

Постановка задачі

Створити **функцію**, в якості аргументів якої подаються:

- **вказівник на початок** списку, а також
- **значення**, яке необхідно **знайти**.

Якщо такий елемент знайдено, то

- повертається **вказівник** на знайдений елемент;
- в іншому випадку - **NULL**

```
list* Search (list* head, int a)
```

Функція пошуку елемента

```
list* Search(list* head, int a)
{
    list* lst = new list;
    lst = head;
    lst->value = head->value;
    lst->next = head->next;
    for(int i = 0; lst != null; lst = lst->next)
    {
        if(lst->value == a)
            return (lst);
    }
    return(NULL);
}
```