Обектно ориентирано програмиране

КЛАСОВЕ

КАНОНИЧНО ПРЕДСТАВЯНЕ. СТАТИЧНИ ЧЛЕНОВЕ НА КЛАС.

Канонично представяне

Правило на голямата четворка:

- Конструктор по подразбиране
- Деструктор
- Конструктор за копиране
- Оператор за присвояване

Защо е голяма четворката?

Всички тези функции се използват в стандартни ситуации

- Конструктор по подразбиране
  - при инициализация без указване на конкретен конструктор
  - при инициализация на динамичен масив
- Конструктор за копиране
  - при инициализация на обект с друг
  - при предаване на параметри към функции
  - при връщане на резултат от функции
- Оператор за присвояване
  - при копиране след инициализация
- Деструктор
  - при унищожаване на обекта

Кога пишем голямата четворка?

Когато обектът трябва да управлява външни за него ресурси.
Най-често: когато обектът работи с динамична памет

- Конструктор по подразбиране
  - заделяме минимална памет или установяваме указателя в nullptr
- Конструктор за копиране
  - заделяме същото количество памет като при оригинала
  - прехвърляме данните
- Оператор за присвояване
  - освобождаваме заетата памет
  - заделяме същото количество памет като при оригинала
  - прехвърляме данните
- Деструктор
  - освобождаваме заетата памет

Кога пишем голямата четворка?

Когато обектът трябва да управлява външни за него ресурси.

Най-често: когато обектът работи с динамична памет

- Конструктор по подразбиране
  - заделяме минимална памет или установяваме указателя в nullptr
- Конструктор за копиране
  - заделяме същото количество памет като при оригинала
  - прехвърляме данните
- Оператор за присвояване
  - освобождаваме заетата памет
  - заделяме същото количество памет като при оригинала
  - прехвърляме данните
- Деструктор
  - освобождаваме заетата памет

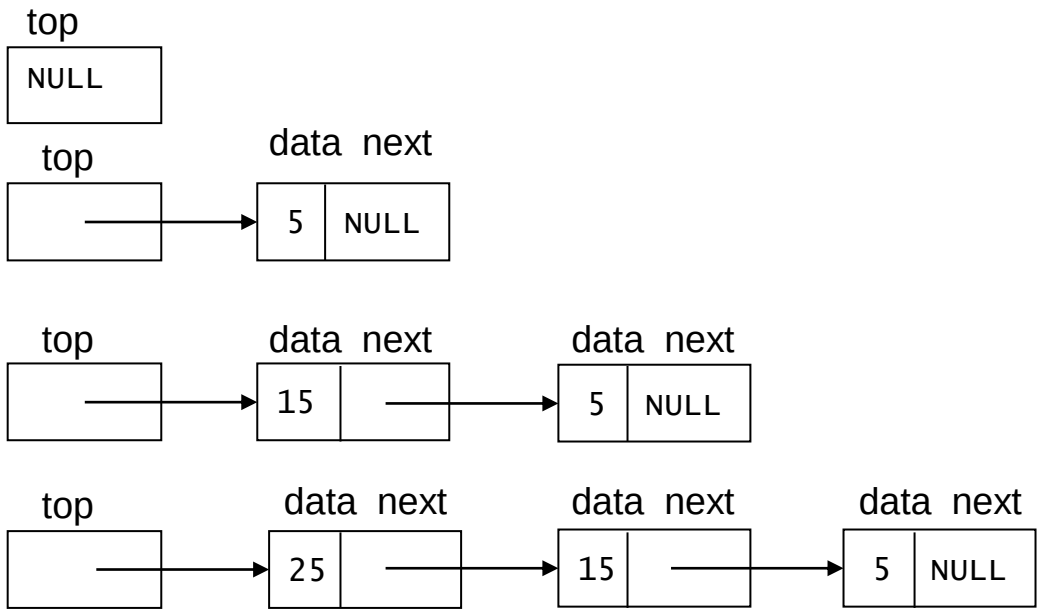
Приложение на средствата за работа с динамичната памет

Ще приведем в канонично представяне:

- Нарастващ стек
- Свързан стек

Приложение на средствата за работа с динамичната памет

Ще конструираме клас `LinkedStack`, който ще реализира свързаното представяне на стек от цели числа.



Приложение на средствата за работа с динамичната памет ...

Забелязваме, че има указател top, който в първия случай представя празен стек, а в останалите случай – непразен, като сочи двойна кутия с информационна част (data) от тип int и свързваща част (next) от типа на top. Това представяне ще реализираме по следния начин:

```
struct StackElement {
    int data;
    StackElement* next;
};
```

Приложение на средствата за работа с динамичната памет …

След тази дефиниция top представя празен стек. Включването на елемента 5 можем да направим чрез изпълнение на следните действия:

```
StackElement *top = NULL, *p;
p = top;
top = new StackElement;
top->data = 5;
top->next = p;
```

Включването на 15 ще направим по аналогичен начин

```
p = top;
top = new StackElement;
top->data = 15;
top->next = p;
```

Приложение на средствата за работа с динамичната памет …

а на 25 – чрез

```
p = top;
top = new StackElement;
top->data = 25;
top->next = p;
```

Тези разсъждения показват, че който и да е елемент x може да се **включи** в стека чрез изпълнение на фрагмента:

```
p = top;
top = new StackElement;
top->data = x;
top->next = p;
```

Приложение на средствата за работа с динамичната памет …

Изключването на елемент от последния стек води до получаване на стека, илюстриран на по-горната стъпка на същата фигура и може да се реализира така:

```
int x;
p = top;
x = top->data;
top = top->next;
delete p;
```

В x е запомнен изключеният елемент.

Приложение на средствата за работа с динамичната памет …

```
struct StackElement {
   int data;
   StackElement* next;
};

class LinkedStack {
private:
   StackElement* top;
   void copyStack(LinkedStack const&);
   void deleteStack();
public:
   // създаване на празен стек
   LinkedStack();
   // конструктор за копиране
   LinkedStack(LinkedStack const&);

   ~LinkedStack();
```

Приложение на средствата за работа с динамичната памет …

```
    // Оператор =
    LinkedStack& operator=(LinkedStack const &);
    // селектори
    // проверка дали стек е празен
    bool empty() const;
    // намиране на елемента на върха на стека
    int peek() const;
    // мутатори
    // включване на елемент
    void push(int);
    // изключване на елемент
     int pop();
};
```

Статични членове на клас

В C++ може да дефинираме статични членове (член-данни и член-функции) използвайки ключовата дума **static**.

Когато декларираме член-данна на клас като статична, това означава, че независимо колко обекта на класа са създадени, съществува само едно копие на статичната член-данна.

Статичните член-данни се споделят от всички обекти на класа. Памет за тях се заделя в статичната памет.

Статичните член данни могат да се инициализират само извън класа.

Пример: box.cpp

Статични членове на клас

Като се декларира член-функция като статична, това я прави независима от обектите на класа. Статична член-функция може да бъде извикана дори да не съществуват обекти на класа. Те се извикват чрез пълното име

<име на клас>::<име на статична член-функция>(<параметри>);

Имат достъп само до статични член-данни и член-функции.

Указателят **this** не се предава на статичните член функции.

Пример: box2.cpp