Теми 20-24
Програмни Езици

Алекс Иванов Цветанов

38гр, КСИ, ФКСТ

2024-2025

Тема 21. Заделяне на обекти от динамичната памет. Проблеми, породени от взаимодействията между обекти.

Динамична памет

Памет, която се заделя от програмата, в следствие на нейното изпълнение

```c
#include <stdlib.h>

int* ptr = (int*)malloc(sizeof(int)); // Заделяне на памет за един int

if (ptr != NULL) {
    *ptr = 42; // Използване на заделената памет
}

// Освобождаване на паметта
free(ptr);
ptr = NULL; // Зануляване на указателя
```

```c
int* ptr = new int; // Заделяне на памет за един int
*ptr = 42; // Използване на заделената памет
// Освобождаване на паметта
delete ptr;
ptr = NULL; // Зануляване на указателя
```

```c
#include <stdlib.h>

int* arr = (int*)malloc(10 * sizeof(int)); // Заделяне на масив от 10 елемента

if (arr != NULL) {
    for (int i = 0; i < 10; ++i) {
        arr[i] = i; // Използване на масива
    }
}

// Освобождаване на паметта за масива
free(arr);
arr = NULL; // Занулляване на указателя
```

```cpp
int* arr = new int[10]; // Заделяне на масив от 10 елемента

for (int i = 0; i < 10; ++i) {
    arr[i] = i; // Използване на масива
}

// Освобождаване на паметта за масива
delete[] arr;
arr = NULL; // Занулляване на указателя
```

```c
#include <stdlib.h>

typedef struct {
    int id;
    char* name;
} Person;

Person* createPerson(int id, const char* name) {
    Person* p = (Person*)malloc(sizeof(Person));
    if (p != NULL) {
        p->id = id;
        p->name = (char*)malloc(strlen(name) + 1);
        if (p->name != NULL) {
            strcpy(p->name, name);
        } else {
            free(p);
            p = NULL;
        }
    }
    return p;
}

void freePerson(Person* p) {
    if (p != NULL) {
        free(p->name);
        free(p);
    }
}

Person* p = createPerson(0, "Pesho");
freePerson(p);
```

```cpp
struct Person {
    int id;
    char* name;
    Person(int id, const char* name) {
        this->id = id;
        this->name = new char[strlen(name) + 1];
        strcpy(this->name, name);
    }
    ~Person() {
        delete[] name;
    }
};

Person* p = new Person(0, "Pesho");
delete p;
```

Тема 22. Приятелски класове и приятелски функции. Статични членове на клас.

Приятелски класове и приятелски функции

- Ако `class A` маркира друг клас и/или метод (функция) като „приятелски”, то той има достъп до всеки член-променлива и член-фукция без значение от private, public, protected.

```
class A
{
        ...
        friend class B;
        friend int get_sum(A obj);
};
```

```
class B
{
        ...
};
```

```
int get_sum(A
obj)
{
        ...
}
```

```
public:
int a, b;
int sum() const { return a+b; }
```

```
private:
int a, b;
int sum() const {return a+b; }
```

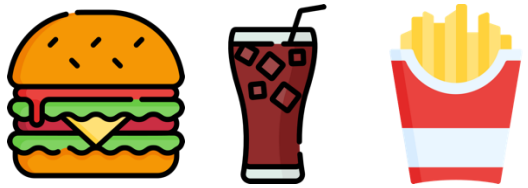
```
protected:
int a, b;
int sum() const {return a+b; }
```

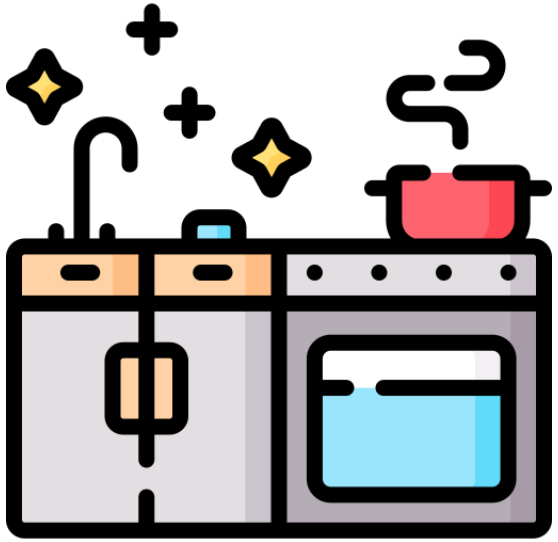
```
A obj1, obj2;
int sum() { return obj1.sum() + b.sum();
}
int sum() {
   return obj1.a + obj1.b + obj2.a +
obj2.b;
}
```

```
return obj.a+obj.b;
return obj.sum();
```

Design Pattern: Facade & Builder

Приложение: McDonald’s app



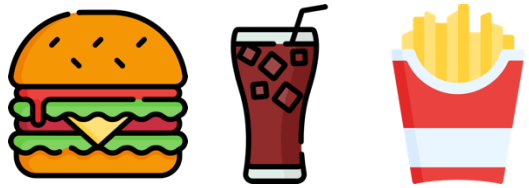


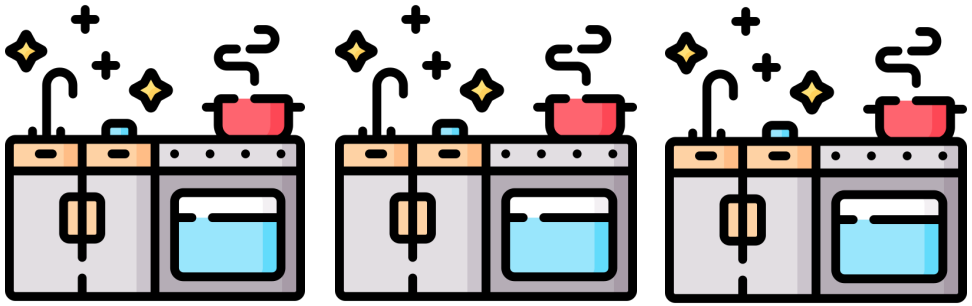
Статични членове на клас

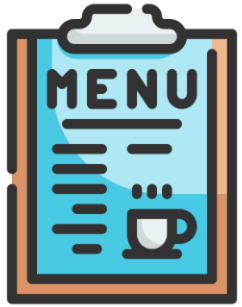
- Това са членове (променливи/функции), чиито стойност/имплементация НЕ зависят от наличието на инстанция (this).
- Тези членове са уникални и независимо колко пъти и от къде ги извикаме, те ще са си на същите адреси.

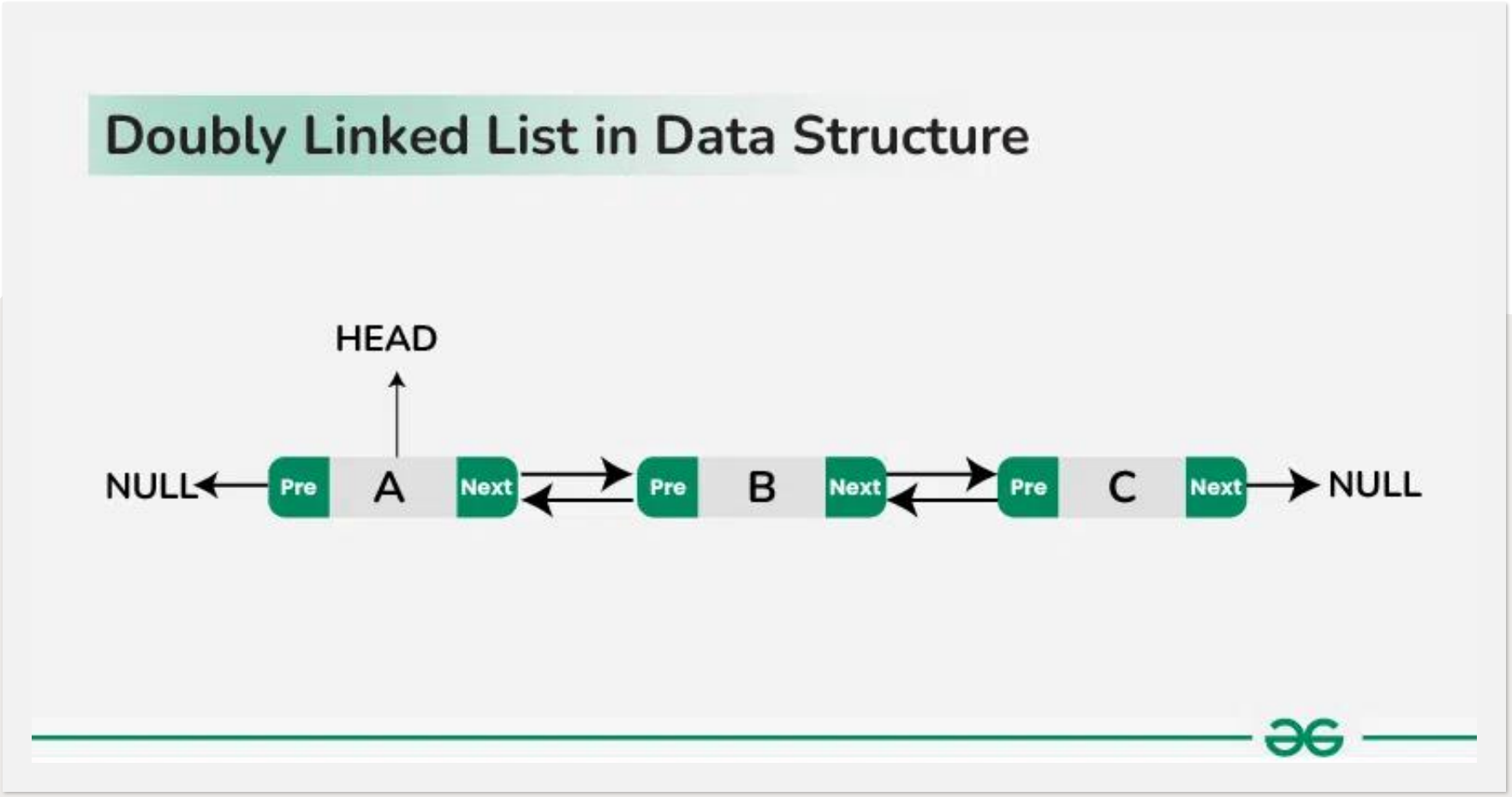
Design Pattern: Singleton

Приложение: McDonald’s app






MENU



Двусвързан списък

Конструиране на вградени обекти. Деструкция на вградени обекти.

class B в class A

public, private, protected имат същото значение

class B може да е public, private или protected

Overloaded operators

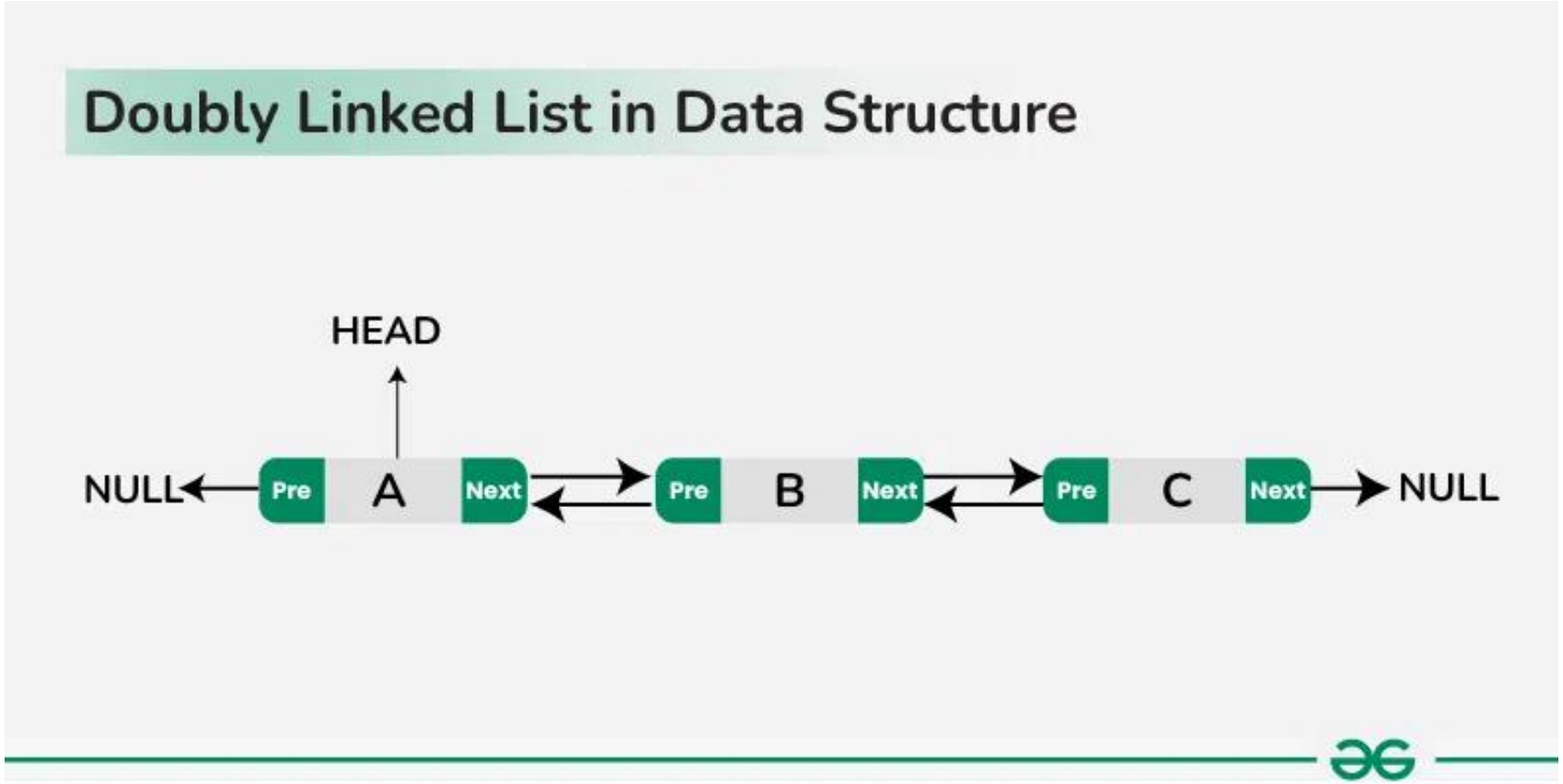
When an operator appears in an expression, and at least one of its operands has a class type or an enumeration type, then overload resolution is used to determine the user-defined function to be called among all the functions whose signatures match the following:

Expression	As member function	As non-member function	Example
@a	(a).operator@ ( )	operator@ (a)	`!std::cin` calls `std::cin.operator!()`
a@b	(a).operator@ (b)	operator@ (a, b)	`std::cout << 42` calls `std::cout.operator<<(42)`
a=b	(a).operator= (b)	cannot be non-member	Given `std::string s;`, `s = "abc";` calls `s.operator=("abc")`
a(b...)	(a).operator()(b...)	cannot be non-member	Given `std::random_device r;`, `auto n = r();` calls `r.operator()()`
a[b...]	(a).operator[](b...)	cannot be non-member	Given `std::map<int, int> m;`, `m[1] = 2;` calls `m.operator[](1)`
a->	(a).operator-> ( )	cannot be non-member	Given `std::unique_ptr<S> p;`, `p->bar()` calls `p.operator->()`
a@	(a).operator@ (0)	operator@ (a, 0)	Given `std::vector<int>::iterator i;`, `i++` calls `i.operator++(0)`
In this table, `@` is a placeholder representing all matching operators: all prefix operators in @a, all postfix operators other than -> in a@, all infix operators other than = in a@b.			

In addition, for comparison operators `==`, `!=`, `<`, `>`, `<=`, `>=`, `<=>`, overload resolution also considers the *rewritten candidates* generated from `operator==` or `operator<=>`. (since C++20)

Note: for overloading `co_await` (since C++20) user-defined conversion functions, user-defined literals, allocation and deallocation see their respective articles.

Припокриване на оператори. Същност, ограничения. Припокриване на аритметични операции.



Двусвързан списък

Преобразувания и операции - преобразувания.

Live Demo

Благодаря за вниманието!