

# Упражнение 10

## Указатели и динамична памет

### Видове памет.

- **Регистрова памет** - повече за нея по КАРХ.
- **Статична памет** - статични и глобални променливи.
- **Автоматична (стекова) памет** - локални променливи.
- **Динамична памет** - част от RAM-та, поисква се от ОС

### Указатели.

Тип данни, който пази адреса и типа към който “сочи” адреса.

// Забележка: Винаги дефинирайте указателите, никога не ги оставяйте без стойност!

// int \* ptr; - не е ОК!

// int \* ptr = nullptr; - ОК.

// int \* ptr = &num; - ОК. (където num е променлива от тип int)

// Забележка 2: Оператора &, преди променлива, връща адреса в паметта на тази променлива.

// Забележка 3: nullptr е “нулев” адрес, за който знаем, че е невалиден, т.е. можем лесно да проверим дали някой указател е валиден чрез проверката ptr == nullptr.

// Забележка 4: Вместо nullptr можем да ползваме NULL.

### Операции с указатели.

Примерен указател `int * ptr = &num;`

- \*ptr - взема **стойността**, запазена на адреса записан в ptr.
- ptr[<индекс>] - “прескача” **индекс по размера на типа** на брой байта напред в паметта и връща стойността на този адрес.
- ptr + <число> - връща адрес, който е с **число по размера на типа** на брой байта по-напред в паметта от ptr.
- ptr - <число> - връща адрес, който е с **число по размера на типа** на брой байта по-назад в паметта от ptr.
- \*(ptr + <число>) е еквивалентно на ptr[<число>].
- Всички сравнения ==, !=, <=, >= се поддържат от указателите.

## Заделяне на динамична памет.

- Стъпки:
  1. Поискване.
  2. Проверка.
  3. Ползване.
  4. Връщане.
- Оператор `new`  
Обръщение към операционната система (поискване на памет).  
Може да хвърли грешка, ако няма достатъчно памет.
- `std::nothrow`  
Казва на ОС да не хвърля грешка ако няма памет, а да върне `nullptr` (или `NULL`).

Примери:

// Заделя памет за 1 цяло число в динамичната памет.

```
int* pNum = new (std::nothrow) int;
```

// Заделя памет за 10 цели числа в динамичната памет.

```
int* pArr = new (std::nothrow) int[10];
```

## Проверка за динамична памет.

// От горните примери:

```
if (pNum == nullptr) {  
    std::cout << "Not enough memory!";  
    return 1;  
}
```

```
if (!pArr) { // !pArr е еквивалентно на pArr == nullptr  
    ...  
}
```

## Ползване на динамична памет.

- Чрез оператора `*`, който връща стойността на заделената памет.  
`*pNum = 5;` // Задава стойност на заделената памет.
- Чрез оператора `[]`, който се ползва както при масив.  
`pArr[3] = 42;` // Записва 42 на 3-тата клетка в паметта след `pArr`.

## Връщане (изтриване) на динамична памет.

Много **важна** стъпка, която не бива да забравяте! Връща поисканата памет от операционната система.

- Оператори `delete` и `delete[]`

// От горните примери:

// Изтрива заделената памет за 1 елемент

`delete pNum;`

// Изтрива заделената памет за повече на брой елементи.

`delete[] pArr;`