Преговор на УП (struct, functions, pointers, dynamic memory)

Struct

Какво е структура ?

Понякога се налага да обединим няколко различни данни в една, с цел по-удобно ползване.

Пример:

```
struct Fruit
{
    char name[20]; //Име на плодът
    int calories; //Калории
    ...
    double weight;
}
```

Function

Какво е функция ?

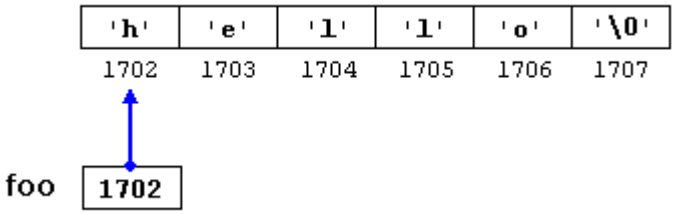
Функция е парче код, което има за цел да свърши някаква работа и да върне резултат от тази работа (не е задължително да се върне резултат.) Може да се представя като "черна кутия", вършеща дадена работа.

Пример:

```
int calculateCaloriesPerGram (Fruit food)
{
    int result = calories/weight;
    return result;
}
```

Тип указател, оператор *, оператор &

Как са представени променливите в паметта ?



Паметта е разделена на клетки, и всяка клетка си има адрес (пример: 0x7ffd86f053dc).

Как можем да видим адреса на който е записана някоя променлива ?

Чрез оператора **&** (псевдоним/reference)

```
int a = 5;
std::cout << &a;
```

Горният пример ще изведе на екрана някакъв адрес, примерно: 0x7ffe8e5363ec

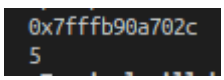
Какво е указател ?

Указателят е специална променлива, чиято стойност е адрес в паметта

```
int a = 5;
int* pointerToA = &a;

std::cout << pointerToA << std::endl;
std::cout << *pointerToA;
```

Този пример, отново ще изведе адреса на **a** в паметта, а след това, ще изведе стойността на **a**.



Операторът ***** поставен пред името на променлива от тип указател, връща стойността, която е записана на съответният адрес

Защо са ни нужни указатели ?

Указателите са много важна част от езика C++ (а и като цяло в програмирането). Когато се подаде променлива на някаква функция, функцията работи с копие на променливата, а не директно с променливата.

Пример:

```
#include <iostream>

void addOneToA(int a)
{
    a = a + 1;
}
int main()
{
    int a = 5;
    int* pointerToA = &a;
```

```
    std::cout << "Before addOneToA() " << a << std::endl;
    addOneToA(a);
    std::cout << "After addOneToA() " << a;
    return 0;
}
```

```
Before addOneToA() 5
After addOneToA() 5
```

Но

```
#include <iostream>

void addOneToA(int* somePointer)
{
    *somePointer = *somePointer + 1;
}
int main()
{
    int a = 5;
    int* pointerToA = &a;

    std::cout << "Before addOneToA() " << a << std::endl;
    addOneToA(pointerToA);
    std::cout << "After addOneToA() " << a;
    return 0;
}
```

```
Before addOneToA() 5
After addOneToA() 6
```

Обяснение на примера: Вместо да подадем на `addToOne()` `int a`, ние му подадохме указател `int* a`, който е свързан към нашата променлива **a** (`int* pointerToA = &a`)

Dynamic memory

Друга полза от указателите има, когато искаме да запишем нещо в динамичната памет.

Какво е динамична памет ?

Разпределението на паметта в една програма на C++ е следното:

- Кодов сегмент - тук се намира кодът на програмта
- Сегмент за данни (инициализиран и неинициализиран) - глобалните и статични променливи
- Стеков сегмент - функции, променливи в тези функции (включително и `main()`), масиви с фиксирана дължина
- Хийп сегмент (динамична памет) - динамични типове данни

Какво са динамичните типове данни ?

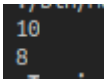
На кратко, това са типове данни, чийто размер не е фиксиран в кода, а по време на изпълнение. Тази памет се достъпва чрез адреси, т.е. са ни нужни указатели, и за нейното заделяне се използват операторите `new` и `delete`. Така можем да заделим масив с размер, дефиниран от потребителя.

Пример

```
int userDefinedArraySize = 5;
std::cin >> userDefinedArraySize;

int* dynamicArray = new int[userDefinedArraySize];

dynamicArray[8] = 8;
std::cout << dynamicArray[8];
```



Примерна задача

Напишете програма, която приема на входа три аргумента - числата *m*, *n*, *a*. Програмата да принтира матрица с размер (m*n), чийто елементи са зададени по следният начин: `Aij = {0 | i != j}; Aij = {a | i == j}`

Решение:

```
#include <iostream>

int main()
{
    int m, n, a;

    std::cin >> m >> n >> a;

    int** matrix = new int*[m]; //Заделяме памет за редовете
    for(int i = 0; i < n; i++)
    {
        matrix[i] = new int[n]; //Заделяме памет, като всеки "ред" е нов масив
    }

    for(int i = 0; i < m; i++)
    {
        for(int j = 0; j < n; j++)
        {
            if(i == j) matrix[i][j] = a; //Ако i == j, Aij = a;
            else matrix[i][j] = 0; //Ако i != j; Aij = 0;
        }
    }

    //Принтираме матрицата
    for(int i = 0; i < m; i++)
    {
        for(int j = 0; j < n; j++)
        {
            std::cout << matrix[i][j] << " ";
        }
```

```
            std::cout << std::endl;
        }
        return 0;
    }
```
