# **1. Указатели и указателна аритметика.**

Указателите в C++ са променливи, които съхраняват адреси на паметта. Те осигуряват начин за непряк достъп до местата в паметта, което позволява динамично разпределяне и манипулиране на паметта. Аритметиката на указатели включва извършване на аритметични операции върху указатели, което се използва предимно за навигация по масиви и динамично разпределена памет.

При аритметиката на указатели добавянето на целочислена стойност към указател го увеличава с толкова байта, в зависимост от размера на типа данни, към който той сочи. Например добавянето на 1 към указател на цяло число увеличава указателя с размера на цяло число.

По същия начин изваждането на целочислена стойност от указател го намалява с толкова байта. Аритметиката на указателите може да се използва и за навигация в масиви, тъй като увеличаването на указател към елемент на масив го премества към следващия елемент на масива.

Адресна аритметика с указатели:

p+i = p+i\*sizeof(T) за T\* p; където sizeof е оператор връщащ брой байтове отделен за типа Т. Също +,-,++,--,==,!=,<,>,<=,>= стават за адресна аритметика.

Т\* const a; //константен указател (change \*a, but not a)

const T\* a; //указател сочещ към константа (change a but not \*a)

const T\* const a; //константен указател към константа

# **2. Едномерни и многомерни масиви. Основни операции с масиви – индексиране. Сортиране и търсене в едномерен масив – основни алгоритми.**

В С++ връзката между масиви и указатели е голяма. Тя се състои в това, че масивите са указатели към техните „първи“ елементи. Това позволява да се разглеждат указателите като алтернативен подход за обхождане на масив. Иначе масивът е статично или динамично заделени n на брой клетки с размерност sizeof(T) в ОП(оперативната памет). Ако е статично, то клетките са последователни в паметта и още по време на компилиране се заделя паметта за тях и може с указател да обхождаме масива. При динамичните клетките са разпръснати по паметта и по време на изпълнение на програмата се случва манипулацията с ОП и не става да бъдат обхождани с указател тъй като не са последователни клетки в паметта. Нека имаме int a[100].

a[0] <-> \*a; a[1] <-> \*(a+1); a[i] <-> \*(a+i);

Имената на масивите са константни указатели, така че ++,-- и присвояване на стойност не са приложими.

За 2D масив – int b[10][20]. b е константен указател към първия елемент на едномернищ масив b[0], b[1], …, b[9], като всеки от тях е константен указател към b[i][0], i in {0, …, 9}

\*\*b <-> b[0][0]; \*b <-> b[0]; \*(b+1) <-> b[1]; \*(b+i) <-> b[i]; \*(b+i)[j] <-> b[i][j]; \*(\*(b+i)+j) <-> b[i][j]

# **3. Рекурсия – пряка и косвена рекурсия, линейна и разклонена рекурсия.**

Един обект е рекурсивен, ако съдържа себе си или е дефиниран чрез себе си. Пример: n!

С++ позволява дефинирането на структури с рекурия и рекурсивни функции. Ще разгледаме само рекурсивни функции. В тялото на функцията може да бъде извикано друго вече дефинирано или декларирана функция, а може и тя да извиква сама себе си.

Функция е косвено рекурсивна, ако F1 -> F2 -> F3 -> … -> Fn -> F1

Функция е пряко рекурсивна, ако в тялото си се извиква.

Пример:  
int factorial(int n) {

If (n==0) return 1;

else return n\*factorial(n-1);

}

int fib(int n) {

if (n == 0 || n == 1) return 1;

else return fib(n-2) + fib(n-1);

}

factorial е пример за линейна рекурсия, а fib е пример за разклонена рекурсия. В случая се пораждат две независими извиквания на fib и сложността е O(2^n).

Хубаво е, ако съществува ефективно итеративно решение за проблем, тогава да не се използва рекурсия. Рекурсията е удобно за работа с рекурсивни структури, обхождане на графи и т.н.