# **1. Принципи на структурното програмиране.**

Основните принципи са 2 – принцип за модулност и принцип за абстракция на данните

- принцип за модулност – програмата се разделя на подходящи части с висока кохезия, всяка от които се реализира чрез определени средства

- принцип за абстракция на данните – методите за използване на данните се отделят от методите за тяхното конкретно представяне. Така програмата работи с неуточнено представяне на данните.

# **2. Управление на изчислителния процес. Основни управляващи конструкции – условни оператори, оператори за цикъл.**

**2.1 Управление на изчислителния процес**

Специално в С++ изпълнението на срр програма започва от main функцията. От своя страна тя може да извиква други и нормалното изпълнение на програмата завършва с изпълнението на main. Най-общото разпределение на оперативната памет за изпълнение на програма на С++ (зависи от ОС):

- програмен код – записан изпълним код на програма

- област на статичните данни – записани глобални обекти

- област на динамичните данни – динамични структури от данни които могат да се освобождават и заделят по време на изпълнение на програмата

- програмен стек – съхранява блокове от памет/стекови рамки

int main(int argc, char\* argv[]) {}

**2.2 Основни управляващи конструкции**

Основните оператори в С++ са:  
 - оператор за присвояване

- оператор за условен преход - if, if else, switch (оператор за

- оператор за безусловен преход – goto

- операторите за цикли – for, while, do-while

- оператор break – излизане от цикъл или switch

- оператор continue – безусловно прескача към следващата итерация на цикъл

# **3. Променливи – видове: локални променливи, глобални променливи; инициализация на променлива; оператор за присвояване.**

Променливата е място за съхранение на данни, което може да съдържа различни стойности от някакъв определен тип. Идентифицира се със зададено от потребителя име. Типът може да е вграден в езика или АД (абстракция на данни – class, struct) зададен от програмиста. В С++ има вградени скаларни: bool, int, double(реални), char(символни), enum(изброим), \*(указател), &(псевдоним), а съставните са array(масив), vector(вектор), запис?. Нека имаме T-тип, с Т\* дефинираме указател към Т, а с Т& дефинираме псевдоним на Т. Dom(T\*) = адреси на променливи с тип Т заедно с nullptr, а Dom(T&) = всички вече дефинирани променливи от тип Т. Дефиниция с инициализация на Т& е задължително и не се сменя. Ако имаме Т a; с &a извличаме адреса на променливата а. Ако имаме Т\*а, то с \*а извличаме съдържанието на клетката, към която сочи указателя а.

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

Променливите могат да имат глобално, локално или област на клас действие. Глобалните могат да се ползват от всички функции и променливи на програмата. Дефинициите са извън всички функции и са валидни от реда на дефиницията до края на програмата. Локалните имат област на действие реда на дефиницият им до края на тази функция. Условен оператор,оператор за цикъл – имат block, scope в С++. Използването на глобални променливи се счита за лоша практика. Има правило, че локално дефинираните променливи със същото име като някои променливи на по-висока област на действие я скрива. Скаларните типове променливи се инициализират чрез оператора за присвояване „=“. int a = 5; int c = 5+3\*a;

Той предизвиква оценка на дясната си част и след това я присвоява тази оценка на левия си аргумент. Ако са различни типове се опитва да ги преобразува иначе програмата гърми с грешка.

# **4. Функции и процедури. Параметри – видове параметри. Предаване на параметри – по име и по стойност. Типове и проверка за съответствие на тип**

Функцията е самостоятелен фрагмент на програмата – отделна програмна единица, съдържаща описание на променливи и набор от оператори на езика. Те се затварят между фигурни скоби и се наричат тяло на функцията. Функцията има възможност да предава и получава информация към и от други функции. За да се предаде в извиканата функцията стойност на една променлива е необходимо тази променлива да е включена в списъка на предаваните стойности (списък на аргументи) Обикновено след изпълнението на дадена функция в извикващата функция се връща резултат от определен тип от някаква изчисления. Ако една функция няма списък с аргументи и не връща резултат, то нея я наричаме процедура. Ако не връща нищо типът ѝ е void и може да се изпусне return.

int gcd(int a, int b) {

if (b == 0) return a;

return gcd(b, a%b);

}

В примера параметрите а и b на функцията gcd се наричат формални параметри на функцията. В момента в който бъде извикана функцията с конкретни параметри, още се наричат фактически параметри, тогава се свързват формалните с фактическите и се изпълнява тялото на функцията (още заделя се нова стекова рамка, в която се пазят return адрес, както и всички данни в текущата функция на върха на стека)

За предния пример свързване на формалните с фактическите е пример за call by value, защото само се копират стойностите на фактическите параметри и изчисленията в gcd не влияят на оригиналните променливи.

Следващият пример е за call by name, тук ако извикаме swap(&x,&y) то след извикването \*x и \*y ще имат разменени стойности. Може и чрез псевдоними (в С++) да направим call by name. По-специалното за псевдонимите е, че не се заделя памет в стековата рамка при обръщането към функцията както при указателите, а параметърът се „закача“ за x както се „закача за y.

void swap(int\* a; int\* b) {

int tmp = \*a;

\*a = \*b;

\*b = temp;

}

С++ е строго типизиран език. По време на компилация се прави проверка за съответствитео на типове, както на списъка от аргументи, така и на типа на резултата на извикваната функция. Ако бъде открито несоътветствие между фактическите типове и типовете, декларирани в прототипа на функцията, ще бъде приложено неявно конвертиране (implicit casting), ако е възможно. Ако не е възможно ще се получи грешка по време на компилация. Прототипът на функцията предлага информация на компилатора за типовете за които проверява.

# **5. Символни низове. Представяне в паметта. Основни операции със символни низове.**

В C++ символните низове обикновено се представят като масиви от символи, завършващи с нулев символ (’\0‘). Този нулев символ означава край на низа. Низовете могат да се манипулират с помощта на различни библиотечни функции, предоставени от стандартната библиотека (STL), или с помощта на традиционните функции за стрингове в стил C.

Основните операции с низове в C++ включват конкатенация, сравнение, копиране и извличане на поднизове. Тези операции могат да се извършват с помощта на вградени оператори или библиотечни функции, като + за конкатенация, == за сравнение, strcpy() за копиране и substr() за извличане на поднизове.