В езика С функцията е еквивалент на понятието подпрограма в езика Фортран, а също така еквивалент на процедура или функция в Паскал. Функцията осигурява удобен начин за капсулация на поредица изчисления, които впоследствие могат да се използуват многократно. Във функционално отношение тя е част от програма, с изпълнението на която се получават определени резултати. Като програмна единица *функцията* се оформя по строго фиксирани правила и представлява **самостоятелен фрагмент от програмата**. Този *фрагмент* може да се изпълнява **мнгократно** в различни части от програмата, като към него се прави обръщение по име.

Тема 8: **Функции в езика С. Въведение – понятие за функция.**

Най-малките единици на които може да се декомпозира една С-програма са функциите. Тя изглежда като линейна структура (фиг.8.1). По-общият синтаксис на функция в езика С е следният:

-------------------------------------------------------------------------------------------------

тип\_на резултата име\_на\_функцията (списък\_на\_параметрите)

описание на типа на параметрите

{

описание на локалните променливи;

оператори;

......

return (израз);

......

оператори;

......

return (израз); Fig.8.1

......

оператори;

}

-----------------------------------------------------------------------------------------

***Функцията*** е *именована логически самостоятелна компонента от потребителcката програма, проектирана за решаване на определена задача, която може да обменя данни с други за нея (външни) програмни единици (функции)*.

Името на главната функция задължително е ***main***, а останалите функции получават имена от програмиста или в зависимост от предназначението си, но имена на валидни *идентификатори*. Функциите в езика С не могат да имат повече от едно име. Функциите са два вида – библиотечни и потребителски. Библиотечните функции са структурирани по функционален признак в отделни header файлове (например – вход/изход – **scanf(..)**, **printf(…)** – във файла **stdio.h**). За тяхното използуване в потребителските програми е достатъчно в тях да се добавят препроцесорните директиви за включване на съответните header файлове, в които се намират декларациите им. Те са (декларациите) достатъчни за С- компилатора при генериране на изпълнимия код на потребителската програма. Свързващия редактор излича изпълнимия им код от библиотеката с изпълнимите кодове (LIB) при създаване на изпълнимия код на потребителската програма (EXE).

Използуването на потребителски функции се налага от необходимостта за многократни еднотипни обработки върху различни множества от входни данни. Например – еднотипната обработка на 3 целочислени масива би увеличило изпълнимия код на потребителската програма с описанието на еднакъв в алгоритмично отношение код върху еднотипни данни. При реализирането на функция за обработка на един масив и последващото й използуване във функцията **main** ще намали програмния код чрез изпълнение на един и същи код (кода на функцията).

Функцията е самостоятелна единица, преработваща постъпилите входни данни (подадени чрез входните параметри) по вложения в нея алгоритъм до определено множество от изходни данни. Ако се използува аналогията от математическа гледна точка функцията може да се раглежда като автомат работещ по закона**: f(arg1,arg2,…)**, т.е функция зависеща от определен брой аргументи. Съгласно стандарта на езика С дефиницията на функцията (пълното описание) се осъществява с формата показан на фиг.8.1, състоящ се от заглавен оператор и тяло на функцията (блок – съставен оператор – {….}):

-----------------------------------------------------------------------------------------

<return\_type> identifier\_function(list\_of\_arguments)

{

декларативна част

операторна част

заглавен оператор

<type1> variable1, ….variableN;

………...

<type2> variable21, ….variable2N;

statement\_1;

statement\_2;

……….

statement\_N; Фиг. 8.1

return value;

}

---------------------------------------------------------------------------------------------

където:

<return\_type> - тип на връщания резултат от функцията при нейното изпълнение;

identifier\_function - идентификатор (име) на функцията;

<list\_of\_arguments> -списък от аргументи(параметри) на функцията:

Забележки:

1. Типът на функцията може да бъде един от предефинираните типове в езика С ( int, float, …) или потребителски тип дефиниран с оператора typedef . Ако потребителската функция не връща резултат то тя се описва с тип void, при което в дефиницията й може да се пропусне оператора return в нейния край.

2. Типът на функцията е незадължителен елемент в заглавния оператор на функцията, т.е може да се пропусне, но тогава функцията се счита за функция, връщаща резултат **int**.

3. Идентификаторът на функцията се подчинява на общите правилата в езика С за запис на идентификатори , т.е. буква, последвана от буква или цифра, без наличие на специални символи.

4. Списъкът от аргументи има вида: **<type> arg1, <type> arg2, ….** Той не е задължителен и зависи от функционалността на проектираната функция – може да е празен или от тип **void**. Той се нарича списък на формалните аргументи ( параметри), които се заменят с фактически в точката на извикването на функцията.

5. Декларативната част (ако е необходима) се намира в началото на блока на функцията и не е задължителна.Тя е последвана от процедурната (операторната) част на функцията, в която се описва алгоритъма за съответната обработка. В тази част не се разрешават нови дефиниции или декларации.

6. Идентификаторът на функцията е особен вид указател ” указател към функция ” . Той съдържа адреса на входната точка на изпълнимия код на функцията.

7. При всяко извикване на функцията компилаторът на С генерира код за предаване на управлението на изчислителния процес с възврат в точката на извикване, а след завършването й управлението се предава към следващия изпълним оператор след точката на извикване.

8. Извикването на функция е операция с най-висш приоритет.

**Дефиниция и декларация на функция**

Потребителските програми обикновено се състоят от определен брой малки по обем потребителски функции, което от една страна осигурява по-добрата читаемост на програмата, а от друга по-лесната и настройка. При използуване на разделно компилиране потребителските функции се оформят в отделни файлове, което улеснява последващото им използуване в други потребителски проекти.

В езика С при описанието на различните програмни обекти (самостоятелни програмни единици, каквито са например променливите, структурите и функциите) се използуват два начина - дефиниция и декларация, между които има съществена концептуална разлика:

- Дефиницията е пълното описание на програмен обект, при обработката на който компилаторът на С заделя памет за съхраняване текущите му стойности по време на достъпа му до него.

- Декларация или прототип на функция е формално описание на програмния обект, който компилаторът на С използува при обработката на потребителската програма в етапа на генериране на кода и синтактическия анализ на изходния код –без заделяне на памет.

Декларацията на функцията, съгласно синтаксиса на езика С се реализира със формата:

<return\_type> indentifier\_function(list\_of\_arguments);

т.е. запис на заглавния оператор на функцията, последван от ;

Декларациите на потребителските функции се записват в началото на блока за декларации съгласно структурата на С-програмата. От прототипа на функцията компилаторът на С извлича необходимата информация за синтактичния разбор и генериране на код в точката на извикването й, дори когато все още не е генерирал кода от дефиницията. Правилото в езика С е, че на етапа на компилация , в точката на извикване на потребителска функция, компилаторът трябва да има информация за нея, чрез декларация или дефиниция. При наличие само на декларации етапа на компилация завършва успешно – без съобщения за грешка, но на следващия етап – свързващото редактиране – свързващия редактор (линк едитора) ще даде съобщение за грешка, тъй като не може да намери междинния код на потребителската функция.

Пример:

Напишете програма, която изчислява броя на секундите в едно денонощие. Изведете резултата на екрана заедно с поясняващ текст

Примерно решение

1. Алгоритъм на задачата: в едно денонощие има 24 часа. Всеки час има 60 минути. Всяка минута има 60 секунди. Следователно общият брой секунди в денонощието ще се изчисли по формулата:

Общ брой секунди = 24 \* 60 \* 60

1. Структурата на програмата използва стандартния шаблон за оформяне на главната функция main – заглавие на **main( ),** тяло и връщане на стойност с **return**. С директивата **include** се включва заглавният файл **stdio.h**, който описва стандартните функции за вход/изход. В случая се използва само функцията **printf( )** за изход.
2. При дефинирането на променливите, използвани в програмата няма някакви особености. Дефинираме една променлива **s** от тип **int**, в която ще запишем изчисления брой секунди.
3. Използваме функцията **printf( )** за извеждане на текст-съобщение и за да отпечатаме изчисления брой секунди.
4. Вижте един съкратен вариант на програмната реализация. Вместо да изчисляваме отделно общия брой секунди и да го записваме в променлива, изчисляването става в израз, който е параметър на функцията printf() и изчислената стойност веднага се печата.

***Примерна реализация на програмата:***

#**include <stdio.h> /\* заглавният файл stdio.h съдържа  
описания за входно-изходните функции \*/  
int main(void)  
{ /\* Пример 3. Тази програма изчислява секундите в едно денонощие \*/  
int s; /\* s е променлива, в която записваме изчисления брой секунди \*/  
s = 24 \* 60 \* 60;  
printf("Общ брой секунди в денонощие = %d",s);  
return 0;  
  
} /\* край на main \*/**

Функцията се извиква чрез името й, като в скобите се задава списък от фактически параметри, разделени със запетая. Съответствието между формалните и фактическите параметри трябва да бъде по брой, тип и място. Функцията се извиква в операция присвояване, например:

**К= isalpha(a);**

или при извеждане, например:

**printf(“k=%d”, isalpha(a));**

Ако функцията не връща резултат за тип на резултата се пише **void** и тази фунция се извиква на отделен ред. Ще илюстрираме тези теоретични постановки с два примера:

Пример 1: Да се напише програма, в която функцията first\_func(int a) да удвоява стойността на променлива от главната програма. Върнатият от функцията резултат да се отпечатва:

**#include <stdio.h>**

**int first\_func(int a);**

**int main()**

**{**

**int k=6;**

**printf(“k=%d”, first\_func(k));**

**return 0;**

**}**

**int first\_func(int a)**

**{**

**a+=a;**

**return a;**

**}**

Същият резултат се получава, ако извеждането на резултата се прави вътре във функцията, а тя самата не връща резултат както е видно от Пример 2, който решава същата задача:

**#include <stdio.h>**

**void first\_func(int a);**

**int main()**

**{**

**int k=6;**

**first\_func(k);**

**return 0;**

**}**

**void first\_func(int a)**

**{**

**a+=a;**

**printf(“k=%d”,a);**

**}**

**Аргументи на функцията main**

Съгласно синтаксиса на езика С програмата е линейна структура от функции, от които задължително е присъствието на функция **main()**. Тя обикновено се нарича *главна функция*, но тя не е с някакъв по-висш приоритет спрямо останалите библиотечни или потребителски функции. Компилаторът на С генерира междинния (обектния) код на програма в реда, в които среща декларациите и дефинициите на функциите, от които е изградена потребителската програма. В последващата обработка свързващият редактор трябва да изгради изпълнимия код на програмата, като осигури връзката между всички функции, от които тя е изградена. При това за правилното свързващо редактиране на него му е необходима информация за входната точка (началото), от която трябва да започне изпълнението на програмата. В този смисъл функцията **main()** е задължителна за всяка С програма - това е началото на изпълнимия код, която я определя като ”главна” функция. Това е функцията към която се предава управлението при зареждането на изпълнимия код от зареждащата програма на ОС. От гледна точка на всяка ОС стартираната потребителска програма е активиран потребителски процес, който в най-общия случай може да бъде извикан (стартиран) от друг процес (наричан родителски), като при това често се налага предаването на съвкупност от параметри за изпълнението на извиканата програма. От друга страна при завършване на изпълнението тя може да върне определена стойност на родителския процес – обикновено това е статуса на завършване, който да бъде обработен от родителския процес, например типа на грешката при обработката на входните данни.

Съгласно езика С, за реализиране на достъпа до външни аргументи на **main(),** се описва като функция от цял тип. Тя се дефинира като функция в два варианта:

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

**{ *оператори;***

***return xxx;***

**}**

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

**{ *оператори;***

***return xxx;***

**}**

където:

**int argc** - брой на аргументите от командния ред, с който е стартирана потребителската програма, включително и името на програмата;

**char \*argv[]** - масив от указатели към **ASCIIZ** низове съдържащ командния ред, с който е стартирана програмата;

**char \*envp[]** - масив от указатели към **ASCIIZ** низове съдържащ глобалните променливи на обкръжението на ОС;

**xxx** - резултат, които се връща на процеса, активирал потребителската програма;

**Глобални и локални променливи в езика С**

В тялото на една функция могат да бъдат дефинирани променливи. Те са определени в тялото на конкретната функция и това е тяхната област на действие. Наричат се вътрешни или локални променливи. В езика С дефинирането трябва да е в началото на тялото на функцията. От това място започва така наречената област на видимост на променливата. Една програма на езика С е съвокупност от функции, като между тях и преди тях могат да се дефинират променливи. Те се наричат външни или глобални и са определени само за тези функции, които са записани след тяхното дефиниране.

Например:

**#include <stdio.h>**

**int l,j; //глобални променливи**

**int calc\_str (int a, int b, int c);**

**int main()**

**{**

**int m,n,q,perim; //локални променливи за главната функция**

**...............................................**

**return 0;**

**}**

**int calc\_str(int a, int b, int c)**

**{**

**int d; // локална променлива за функцията calc\_str()**

**…………………………**

**return (d);**

**}**

Чрез глобалните променливи могат да се обменят данни между функциите. Имената на глобалните променливи трябва да са уникални в програмата. Ако в една функция се среща локална променлива с име, което съвпада с името на глобална променлива, то във функцията може да се използува само локалната променлива.

Пример: Да се напише програма, която илюстрира областите на действие на глобални и локални променливи.

**#include <stdio.h>**

**#include <stdlib.h>**

**int a=3;**

**int prim();**

**void do\_printf();**

**int main()**

**{**

**int b;**

**printf(“Globalna: a=%d\n,a);**

**b=a+5;**

**printf(“Main: b=%\n”,b);**

**prim();**

**printf(“Main a=%d\n”,a);**

**do\_printf();**

**system(“pause”);**

**}**

**int prim()**

**{**

**Int a=-3;**

**a+=16;**

**printf(“Function prim: a=%d\n”,a);**

**}**

**void do\_printf()**

**{**

**printf(“Function do\_printf: a=%d\n”,a);**

**}**

В този пример се дефинира глобална променлива а=3, която се използува само от функциите main() и do\_printf(). Във функцията prim() отново се дефинира променлива със същото име а , която е локална за тази функция. Там тя приема стойност 13, която се извежда. При връщане в главната функция се вижда, че глобалната променлива със същото име не е променила стойността си.

Резултатът е:

**Globalna: a=3**

**Main: b=8**

**Function prim: a=13**

**Main: a=3**

**Function do\_printf: a=3**