УРОК 01

C Programming Tutorial — https://www.unf.edu/~wkloster/2220/ppts/cprogramming_tutorial.pdf
Free C Programming Book — https://books.goalkicker.com/CBook/

1

Добре дошли в Урок 01, Част 01 по програмиране на C ! В този урок ще направим **основен преглед на програмирането на C и неговите функции**.

C е език за програмиране **с общо предназначение, който се използва широко в системното програмиране, вградените системи и разработката на игри**. Разработен е в началото на 70-те години на миналия век от Денис Ричи в Bell Labs като наследник на езика за програмиране B.

Програмите на C се състоят от **поредица от операции**, които се изпълняват **последователно**. **Инструкцията** може да бъде декларация на променлива, присвояване, извикване на функция, условна инструкция или цикъл.

Функцията е блок от код, който изпълнява конкретна задача. Функциите позволяват на програмистите да разбият своя код на по-малки, по-управляеми части, което го прави по-лесен за четене, разбиране и поддръжка. C има вградени функции, като printf( ) и scanf( ), но можете също да създавате свои собствени функции.

Ex_01

Нека да разгледаме пример за **проста C програма**:

C Standard Library - Quick Guide | Tutorialspoint

```
#include <stdio.h>
int main(void) {
printf("Hello, world!\n");
return 0;
}
```

Този програма използва printf( ) функцията за отпечатване на съобщението "Hello, world!" към конзолата. Изявлението return 0 в края на main( ) функцията показва, че програмата е изпълнена успешно.

Ex_02

Сега нека да разгледаме **пример за функция**:

```
#include <stdio.h>
int square(int num) {
return num * num;
}

int main(void) {
int num = 5;
```

```
int result = square(num);
printf("%d squared is %d\n", num, result);
return 0;
}
```

В тази програма сме дефинирали наречена функция square( ), която приема аргумент цяло число и връща квадрата на това число. Също така извикахме square( ) функцията от самата main( ) функция и отпечатахме резултата в конзолата с помощта на printf( ).

Това е всичко за прегледа на програмата и функциите в C.

2

Добре дошли отново в урок 01 от програмирането на C ! В този урок ще разгледаме концепциите за глобални и локални променливи в C.

В C променливите могат да бъдат декларирани или **вътре във функция**, което ги прави локални, или **извън функция**, което ги прави глобални. Нека разгледаме по-отблизо всеки тип променлива.

Локални променливи

Локалните променливи се декларират във функция и са достъпни само в рамките на тази функция. Когато функцията бъде извикана, паметта се разпределя за локалната променлива и когато функцията се върне, тази памет се освобождава.

Ето един пример:

Ex_03

```
#include <stdio.h>
void myFunction() {
int myVar = 5; // Declared inside the function
printf("myVar inside myFunction(): %d\n", myVar);
}

int main(void) {
myFunction();
// printf("myVar outside myFunction(): %d\n", myVar); // This line would cause an error
return 0;
}
```

В тази програма имаме функция, наречена myFunction(), която декларира локална променлива, наречена myVar. Когато myFunction()се извика от main()функцията, стойността на myVarce отпечатва на конзолата. Въпреки това, ако се опитаме да отпечатаме стойността на myVarизвън функцията, ще получим грешка, защото myVarне е достъпна извън функцията.

Глобални променливи

Глобалните променливи, от друга страна, се декларират извън всяка функция и са достъпни за всички функции в програмата. Те получават памет при стартиране на програмата и се освобождават само когато програмата излезе.

Ето един пример:

Ex_04

```
#include <stdio.h>
int myVar = 5; // Declared outside the function
void myFunction() {
printf("myVar inside myFunction(): %d\n", myVar);
}
int main(void) {
myFunction();
printf("myVar outside myFunction(): %d\n", myVar);
return 0;
}
```

В тази програма имаме глобална променлива, наречена, myVar, която е декларирана извън всяка функция. И двете - myFunction( ) и main( ) функции могат да получат достъп до стойността myVar и да я отпечатат на конзолата.

Важно е да се отбележи, че докато глобалните променливи могат да бъдат полезни, те също могат да доведат до грешки и да направят кода по-труден за поддръжка. Като цяло е добра практика да се ограничи използването на глобални променливи и да се предпочитат локалните променливи, когато е възможно.

Това е всичко за глобалните и локалните променливи в C.

3

Добре дошли в част 3 на Урок 01 по програмиране на C! В този урок ще разгледаме **концепцията за ПРЕ-ПРОЦЕСОРА в C**.

Препроцесорът е програма, която се изпълнява преди компилатора и отговаря за обработката на директиви, които започват със символа '#'. Тези директиви не са част от самия език C, но предоставят допълнителна функционалност на кода.

Препроцесорът извършва текстови заместявания, включвания на файлове и условна компилация. Нека разгледаме по-отблизо всяка от тези функции.

Текстови заместявания :

Препроцесорът може да извършва текстови заместявания с помощта на #define директивата. Тази директива дефинира макрос, който може да се използва в цялата програма за заместване на текст.

Ето един пример:

Ex_05

```
#include <stdio.h>
#define PI 3.14159
int main(void) {
double radius = 5.0;
double circumference = 2 * PI * radius;
printf("The circumference of a circle with radius %f is %f\n", radius, circumference);
return 0;
}
```

[C Standard Library - Quick Guide | Tutorialspoint]

В тази програма сме дефинирали макрос, наречен PI с помощта на #define директивата. След това използваме PI в нашето изчисление обиколката на кръг. По време на компилация препроцесорът заменя всички екземпляри на PIсъс стойността 3.14159.

Включване на файлове :

Препроцесорът може да включва други файлове в програмата с помощта на #include директивата. Тази директива казва на препроцесора да вмъкне съдържанието на друг файл в програмата.

Ето един пример:

Ex_06

```
#include <stdio.h>
#include "myHeader.h"
int main() {
printMessage();
return 0;
}
```

В тази програма сме включили /header/заглавен файл, наречен myHeader.h с помощта на #include директивата. Този заглавен файл съдържа наречена функция printMessage( ), която извикваме от самата main( ) функция.

[Header Files in C/C++ and its uses - GeeksforGeeks]

Условна компилация :

Препроцесорът може да извършва условна компилация с помощта на директивите #if, #ifdef и #ifndef. Тези директиви позволяват на програмиста да включва или изключва определени части от код въз основа на стойността на макрос или дали макросът е дефиниран.

Ето един пример:

Ex_06

```
#include <stdio.h>
#define DEBUG
int main() {
#ifdef DEBUG
printf("Debug mode is on\n");
#else
printf("Debug mode is off\n");
#endif
return 0;
}
```

В тази програма сме дефинирали макрос, наречен DEBUG с помощта на #define директивата. След това използваме #ifdef директивата, за да компилираме условно кода, който отпечатва съобщение на конзолата въз основа на това дали DEBUG е дефинирано или не.

Това е всичко за концепцията на препроцесора в C. В някой следващ урок ще разгледаме управляващите структури в C.

4

Добре дошли в част 4 на урок 01 по програмиране на C! В този урок ще разгледаме **процеса на КОМПИЛИРАНЕ и СВЪРЗВАНЕ на програма в C.**

[1] Компилирането и **[2] свързването** са двете основни стъпки при конвертирането на C код в изпълнима програма, която може да се изпълнява на компютър.

Компилиране / Compiling /

Компилирането е процес на преобразуване на изходния код на C в обектен код, който е представяне на програмата на ниско ниво на машинен език, който може да бъде изпълнен от компютъра.

Компилаторът е отговорен за проверката на синтаксиса на кода и генерирането на обектен код, който може да бъде свързан с други обектни файлове за създаване на изпълнима програма.

Ето един пример:

```
#include <stdio.h>
int main() {
printf("Hello, world!\n");
return 0;
}
```

За да компилираме тази програма, можем да използваме **инструмент от командния ред** като **GCC (GNU Compiler Collection)**. Ще отидем до директорията, където е записана програмата, и ще използваме следната команда:

```
gcc program.c -o program
```

Тази команда казва на компилатора да компилира program.c файла и да генерира изпълним файл, наречен program.

Свързване / Linking /

Свързването е **процес на комбиниране на обектни файлове и библиотеки** в една изпълнима програма.

Когато една програма се компилира, **тя се разделя на няколко обектни файла**, всеки от които съдържа част от кода на програмата. **Линкерът е отговорен за комбинирането на тези обектни файлове и свързването на всички необходими библиотеки,** за да създаде един изпълним файл, който може да се изпълнява на компютъра.

Ето един пример:

Да предположим, че имаме два C файла : main.c и functions.c. Файлът main.c съдържа основната функция, докато functions.c файлът съдържа няколко помощни функции, които се използват от основната функция.

Първо ще **компилираме** всеки файл **поотделно**:

```
gcc -c main.c -o main.o
gcc -c functions.c -o functions.o
```

Тези команди генерират **два обектни файла main.o** и **functions.o**.

След това можем да **свържем** тези **обектни файлове** заедно в една **изпълнима програма**:

```
gcc main.o functions.o -lm -o program
```

Тази команда **казва на линкера да комбинира** main.o и functions.o файловете и да генерира изпълним файл, наречен **program**. -lm е 'ключ', който показва, че трябва да бъде включена и библиотеката math.c

Това е всичко за процеса на компилиране и свързване на програма в C

5

Добре дошли в част 5 на урок 01 по програмиране на C! В този урок ще разгледаме **концепцията за МОДУЛИ** в C.

Модулът е самостоятелна единица код, която може да се използва повторно в различни части на програма или в различни програми като цяло. В C модулите се изпълняват като изходни файлове, като всеки файл съдържа набор от свързани функции и данни.

Използването на модули в C позволява по-добра организация на кода и улеснява поддържането и повторното използване на кода. Нека да разгледаме пример, за да видим как модулите работят на практика.

Да предположим, че имаме програма, която трябва да извърши някои математически изчисления. Можем да създадем модул, наречен **my_math.c** да съдържа съответните функции, ето така:

Ex_07.1

```c
// my_math.c
#include <math.h>
#include <my_math.h> // ВКЛЮЧВАМЕ НАШИЯ ХЕДЪР-ФАЙЛ

double square(double num) {
return pow(num, 2);
}
double cube(double num) {
return pow(num, 3);
}
```

Тук сме дефинирали две функции **square( )** и **cube( )**, които извършват математически операции върху дадено число. Включихме и стандартния заглавен файл на библиотеката **math.h**, който съдържа pow функцията, използвана от нашия модул.

За да използваме този модул в друга част на програмата, ще включим файла math.c в нашия изходен код:

Ex_07.2

```c
// my_main.c
#include <stdio.h>
#include <my_math.h> // ВКЛЮЧВАМЕ НАШИЯ ХЕДЪР-ФАЙЛ

int main(void) {

double num = 5.0;
double sq = square(num);
double cu = cube(num);

printf("The square of %.2f is %.2f.\n", num, sq);
printf("The cube of %.2f is %.2f.\n", num, cu);
```

```
return 0;
}
```

Тук сме включили както стандартния заглавен файл на библиотеката, stdio.h така и нашия my_math.h файл в нашия изходен код. След това можем да извикаме функциите square( ) и cube( ) от самата main функция. А така изглежда и самия хедър-файл :

Ex_07.3

```
// my_math.h
```

Хедър файл, който съдържа СИГНАТУРИТЕ на методите - (**прототипите на функциите**).

```
// съдържа само сигнатурите на методите - познато още като прототипиране на функции !
double square(double num);
double cube(double num);
```

var 01 Командата за компилиране може да има и следния вид :

```
gcc my_main.c my_math.c -lm -o program -Wall
// Флагът -lm се използва за свързване на mathбиблиотеката. Wall означава 'all Warnings', т.е. при компилиране ще имаме на екран всички предупреждения. Това не са грешки, които да попречат на програмата да работи.
```

var 02 Или може да има и следния вид :

```
gcc my_main.c my_math.c -lm -o program -Wall -Wextra
```

var 03 Също така командата за компилиране може да има и следния вид, но това ще доведе до появата на всички предупреждения, които ще се интерпретират /третират/ като грешки :

```
gcc my_main.c my_math.c -lm -o program -Wall -Wextra -Werror
```

Когато компилираме използвайки горния ред и стартираме тази програма, ще видим следния резултат:

```
The square of 5.00 is 25.00.
The cube of 5.00 is 125.00.
```

Както можете да видите, използването на модули в C ни позволява да поддържаме свързания код преамизиран и повторно използваем.

6

Добре дошли в урок 01 по програмиране на C! В този урок ще разгледаме как да използваме **интегрирана среда за разработка (IDE) за разработване на C програми.**

IDE е софтуерно приложение, което предоставя цялостна среда за разработка на софтуер. Обикновено включва **редактор на код, компилатор, линкер, програма за отстраняване на грешки и други инструменти**, които правят процеса на разработка по-ефективен.

Има много различни IDE за програмиране на C, но ние ще използваме Visual Studio Code (VS Code) в този урок. VS Code е популярна IDE с отворен код, която е безплатна за изтегляне и използване.

За да започнете с VS Code, първо го изтеглете и инсталирайте от официалния уебсайт. Веднъж инсталиран, отворете VS Code и създайте нов файл, като изберете „Файл“ > „Нов файл“ от менюто.

В новия файл можем да напишем нашия C код, така:

```
#include <stdio.h>
int main() {
printf("Hello, world!\n");
return 0;
}
```

Тук сме написали просто "Здравей, свят!" програма в C. За да компилираме и стартираме тази програма, можем да използваме VS Code Terminal.

За да отворите терминала, изберете "Преглед" > "Терминал" от менюто. В терминала отидете до директорията, където е записан C файлът, като използвате cdкомандата. След това използвайте следната команда, за да компилирате програмата:

```
gcc -o hello_world hello_world.c
```

Тази команда ще компилира hello_world.c файла и ще създаде изпълним файл, наречен hello_world. За да стартирате програмата, просто въведете следната команда:

```
./hello_world
```

Това ще изпълни програмата и ще отпечата "Hello, world!" към конзолата.

Използването на IDE като VS Code може да направи програмирането на C много по-ефективно и рационализирано. Той предоставя инструменти като подчертаване на синтаксис, завършване на код и отстраняване на грешки, които могат да помогнат на разработчиците да пишат по-добър код по-бързо.

7

Добре дошли в урок 01 по програмиране на C! В този урок ще разгледаме как да компилираме, свързваме, изпълняваме и изпълняваме други действия с командния интерпретатор.

Командният интерпретатор, известен също като обвивката, е програма, която предоставя интерфейс на командния ред за взаимодействие с операционната система. В този урок ще използваме обвивката Bash на Unix-базирана система.

За да започнете, отворете терминала и отидете до директорията, където е записан C файлът, като използвате командата cd. След това използвайте следната команда, за да компилирате програмата:

```
gcc -o hello_world hello_world.c
```

Тази команда ще компилира hello_world.cфайла и ще създаде изпълним файл, наречен hello_world. Опцията -оуказва името на изходния файл.

След като програмата е компилирана, можем да я изпълним, като напишем следната команда:

```
./hello_world
```

Това ще изпълни програмата и ще отпечата "Hello, world!" към конзолата.

В допълнение към компилирането и изпълнението на програми, командният интерпретатор предоставя много други полезни функции за работа с файлове и директории. Например, можем да използваме lsкомандата за изброяване на съдържанието на директория, cdкомандата за промяна на директории и mkdirкомандата за създаване на нова директория.

Ето няколко примера:

За да видите съдържанието на текущата директория:

```
ls -l
```

За да преминете към друга директория:

```
cd /path/to/directory
```

За да създадете нова директория:

```
mkdir my_directory
```

В допълнение към тези основни команди, командният интерпретатор предоставя много други мощни функции за работа с операционната система. Като се научим как да използваме командния интерпретатор ефективно, можем да станем много по-ефективни и продуктивни C програмисти.

В заключение, командният интерпретатор е мощен инструмент, който може да се използва за компилиране, свързване, изпълнение и извършване на много други действия с C програми. Като овладеем командния интерпретатор, можем да станем много по-ефективни и продуктивни разработчици.

8

Добре дошли в урок 01 по програмиране на C! В този урок ще разгледаме **как да използваме справочна информация за езика C.**

Като C програмист е важно да имате добро разбиране на синтаксиса, семантиката и функциите на стандартната библиотека на езика. За щастие има много налични ресурси, които да ни помогнат да научим и да използваме тази информация.

Един от най-важните ресурси за програмиране на C е официалната спецификация на езика C, която предоставя изчерпателно и подробно описание на езика. Най-новата версия на спецификацията на езика C е достъпна онлайн в HTML формат и може да бъде достъпна на следната връзка: https://port70.net/~nsz/c/c11/n1570.html

Друг важен ресурс за програмиране на C е документацията на стандартната библиотека, която предоставя подробна информация за различните функции и типове данни, налични в стандартната библиотека. Последната версия на документацията на стандартната библиотека е достъпна онлайн на следната връзка: https://en.cppreference.com/w/c

Важно е да се отбележи, че различните C компилатори и платформи може да имат различни реализации на езика и стандартната библиотека. Следователно винаги е добра идея да се консултирате с документацията, предоставена от вашия конкретен компилатор и платформа, за да сте сигурни, че използвате правилния синтаксис и семантика.

В допълнение към официалната документация има и много онлайн ресурси и форуми на общността, които могат да се използват за задаване на въпроси и учене от други C програмисти. Някои популярни ресурси включват Stack Overflow, C Programming subreddit и C board във форума на CodeGuru.

Сега нека да разгледаме пример за реален код и да видим как можем да използваме справочна информация, за да я разберем по-добре:

```
#include <stdio.h>
int main() {
printf("Hello, world!\n");
return 0;
}
```

В този пример включваме стандартния входно/изходен заглавен файл <stdio.h>и дефинираме mainфункция, която връща целочислена стойност от 0. В рамките на mainфункцията използваме printfфункцията, за да отпечатаме низа "Hello, world!" към конзолата.

Като се консултираме със спецификацията на езика C и документацията на стандартната библиотека, можем да научим повече за синтаксиса и семантиката на тези елементи на програмата. Например, можем да научим, че #includeдирективата се използва за импортиране на заглавни файлове и че printfфункцията се използва за отпечатване на форматиран изход към конзолата.

В заключение, като използваме справочна информация и онлайн ресурси, можем да станем по-ефективни и знаещи C програмисти. Докато продължаваме с този курс, ще изследваме още много аспекти на езика C и стандартната библиотека и ще научим как да използваме тези ресурси, за да пишем мощни и ефективни програми.

9 **Прости примерни програми:** Ето една програма, която приема две числа като вход и извежда тяхната сума:

Ex_09

```c
#include <stdio.h>
int main() {
int num1, num2, sum;
printf("Enter first number: ");
scanf("%d", &num1);
printf("Enter second number: ");
scanf("%d", &num2);
sum = num1 + num2;
printf("Sum of the two numbers: %d", sum);
return 0;
}
```

Нека разбием какво се случва тук.

Първо, включваме стандартната библиотека за вход/изход с #include <stdio.h>. След това дефинираме нашата main()функция, която е входната точка на нашата програма.

Вътре main()декларираме три променливи: num1, num2и sum. Всички те са от тип int, което означава, че са цели числа.

Използваме printf(), за да изведем съобщение до потребителя, като го молим да въведе първото число. След това използваме, scanf()за да прочетем число от потребителя и да го съхраним в num1. Повтаряме този процес за второто число.

След това изчисляваме сумата на двете числа и я съхраняваме в sum. Накрая използваме printf()отново, за да изведем сумата на потребителя.

Ето **още една примерна програма**, която демонстрира използването на цикли:

Ex_10

```
#include <stdio.h>
int main() {
int i;
for (i = 0; i < 10; i++) {
printf("%d\n", i);
}
return 0;
}
```

Този програма използва forцикъл за извеждане на числата от 0 до 9 към конзолата. Нека да разгледаме как работи цикълът.

Ние декларираме променлива iот тип int. След това използваме forцикъл, за да итерираме стойностите iот 0 до 9. Цикълът има три части:

Инициализация:i = 0
Състояние:i < 10
Увеличение:i++

Всеки път, когато цикълът се повтори, ние използваме, printf()за да изведем стойността на іна конзолата, последвана от знак за нов ред.

Надявам се тези примери да ви помогнат да илюстрирате някои от концепциите, които разгледахме в този урок!

Още два **допълнителни примера за C програми**:

Изчислете средната стойност на три числа

Ex_11

```
#include <stdio.h>
int main() {
double num1, num2, num3, average;
printf("Enter the first number: ");
scanf("%lf", &num1);
printf("Enter the second number: ");
scanf("%lf", &num2);
printf("Enter the third number: ");
scanf("%lf", &num3);
average = (num1 + num2 + num3) / 3;
printf("The average of %lf, %lf and %lf is %lf\n", num1, num2, num3, average);
return 0;
```

```
}
```

Тази програма подканва потребителя да въведе три числа, изчислява средната им стойност и я отпечатва на конзолата.

Преобразувайте температурата от Фаренхайт в Целзий

Ex_12

```
#include <stdio.h>
int main() {
double fahrenheit, celsius;
printf("Enter temperature in Fahrenheit: ");
scanf("%lf", &fahrenheit);
celsius = (fahrenheit - 32) * 5 / 9;
printf("%.2lf°F is equal to %.2lf°C\n", fahrenheit, celsius);
return 0;
}
```

Тази програма подканва потребителя да въведе температура във Фаренхайт, преобразува я в Целзий и отпечатва резултата на конзолата.