

Język ANSI C

Pierwsze starcie.

Ostatnia aktualizacja: 3 listopada 2025



- 1972 Dennis Ritchie (Bell Labs., New Jersey), projekt języka C na bazie języka B
- 1973 UNIX, jądro w C, pierwszy przenośny system operacyjny
- 1978 D. Ritchie, Brian Kernighan, „*The C Programming Language*”
- 1983 Bjarne Stroustrup, Język C++
- 1989 Standard ANSI C, standardowe C, „pure C”, C89, C90
- 1999 Standard C99
- 2011 Standard C11
- 2024 Standard C23

```
#include <stdio.h>
#define PI 3.1415
```

Dyrektywy preprocesora

```
int main()
{
```

Funkcja główna

```
    int i;
    float x;
```

Deklaracje zmiennych

```
    i = 10 * PI;
    x = 1.0;
```

Instrukcje programu

```
    return 0;
```

```
}
```

Najkrótszy program w C

```
main() {}
```

Witaj świecie

```
#include<stdio.h>

int main()
{
    puts("Witaj świecie!");
    return 0;
}
```

- mały język ale duże możliwości, ważna rola bibliotek
- **pliki źródłowe** *.c zawierają instrukcje programu
- **pliki nagłówkowe** *.h zawierają deklaracje typów i funkcji, nie zawierają instrukcji
- biblioteki: zbiory typów danych, stałych i funkcji
- biblioteki standardowe, np.: `stdio.h`, `math.h`
- dostęp do pamięci i rejestrów
- zwięzły kod - ale nie wolno przesadzać
- C nie chroni programisty przed nim samym

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Dyrektywy to instrukcje zaczynające się od znaku #.

Nie są słowami języka C, wykonywane są przed właściwą kompilacją.

- `#include` *plik*
dołączenie pliku nagłówkowego zawierającego definicje funkcji, typów i stałych

```
#include <stdio.h>
#include "../plik.h"
```

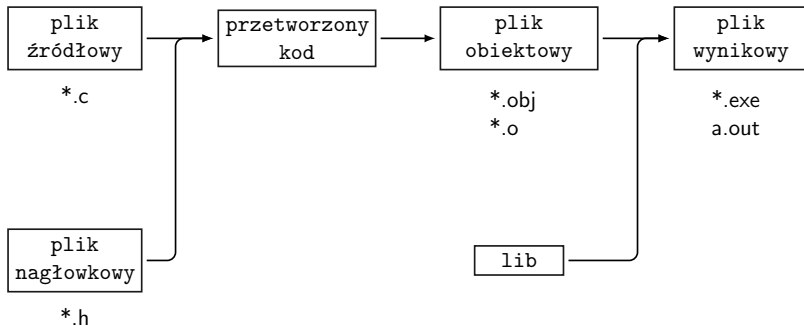
- `#define` *nazwa wartość*
definiuje stałe, tworzy „przezwisek” do słów i wyrażeń (makra)

```
#define PI 3.1415
#define TRUE 1
#define FALSE 0
#define real float
```

1. Preprocessing

2. Kompilacja

3. Linkowanie



Preprocesor wykonuje instrukcje zaczynające się znakiem # (dyrektywy preprocesora). Przygotowuje pliki do kompilacji.

pliki źródłowe (*.c, *.h) $\Rightarrow$ przetworzone pliki

```
#include<stdio.h>
```

```
#define PI 3.14
```

Kompilacja tłumaczy instrukcje C na kod maszynowy
przetworzone pliki $\Rightarrow$ pliki obiektowe (*.obj, *.o)

Konsolidacja (linkowanie) łączy pliki obiektowe w aplikację
pliki obiektowe + biblioteki $\Rightarrow$ program (*.exe , a.out)

- **zmienne** są określone przez typ i unikatową nazwę
- każda zmienna zajmuje pamięć: 1B, 2B, 4B, 8B, ...
- **adres** zmiennej określa jej położenie w pamięci
- binarna reprezentacja zmiennych
- **skończoność** - liczby są reprezentowane poprawnie w pewnym przedziale
- **rozdzielczość** - liczby rzeczywiste są reprezentowane z pewną dokładnością

char a='C';

&a

0	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

 = 67

typ	reprezentacja	zakres wartości
char	znaki (<i>character</i>), kod ASCII	$[-127, 128]$
int	liczby całkowite (<i>integer</i>)	$[-2^{31}, 2^{31} - 1]$
float	liczby rzeczywiste (<i>floating point</i>) reprezentacja zmiennoprzecinkowa najmniejsza dodatnia wartość 1.17×10^{-38}	$[-3.4 \times 10^{38}, +3.4 \times 10^{38}]$
logiczny	brak typu logicznego wartość całkowita 0 to fałsz a wartość różna od 0 to prawda	
void	typ pusty, brak typu	

typ bool wprowadzony w standardzie C23

DEKLARACJA

występuje na początku bloku instrukcji

```
typ identyfikator;  
typ identyfikator1, identyfikator2, identyfikator3;
```

powiązanie zmiennej, stałej lub funkcji danego typu z identyfikatorem (unikatową nazwą)

Przykład:

```
int i;  
float x, y;  
char c, d, e;
```

- dozwolone znaki: litery a-z, A-Z, cyfry 0-9, podkreślnik _
- cyfra nie może być pierwszym znakiem
- małe i duże litery są rozróżniane: a $\neq$ A
- zarezerwowane słowa nie mogą wystąpić w roli identyfikatorów:
 - słowa kluczowe języka (np. while, int, return)
 - identyfikatory zadeklarowane wcześniej (zmiennie, funkcje)

Przykład:

```
int a,b,c;  
float srednica_kola;  
char PewienZnak;  
float x1, x2, x3;  
int _wazna_zmienna;
```

```
int 4abc;  
float średnica;  
char pewien znak;  
int wazna-zmienna;  
float return;
```

INSTRUKCJA PROSTA

wyrażenie ;

```
a = x + 1;
```

Instrukcja prosta jest zawsze zakończona średnikiem.

INSTRUKCJA ZŁOŻONA (BLOK INSTRUKCJI)

```
{  
    instrukcja 1  
    instrukcja 2  
    instrukcja 3  
}
```

```
{  
    float x, y;  
    x = 1.0;    y = 2.4;  
    x = x + y;  
}
```

INSTRUKCJE STERUJĄCE

if else do while for goto

ZAKRES DOSTĘPU ZMIENNEJ

```
int x = 1;
```

zmienna **globalna**,
dostępna w zakresie całego pliku

```
int main()
```

```
{
```

```
    int x = 2;
```

zmienna **lokalna**,
dostępna w zakresie funkcji main,
przysłania zmienną globalną x

```
    {
```

```
        int x = 3;
```

zmienna lokalna
dostępna w zakresie bloku instrukcji {}
przysłania zmienną x z funkcji main

```
    }
```

```
}
```

Zmienne deklaruje się na samym początku bloku (czyli przed pierwszą instrukcją).

Od C99 można deklarować w dowolnym miejscu przed użyciem zmiennej

OPERATORY ARYTMETYCZNE

* / % + -

OPERATORY RELACJI

< > <= >= == !=

OPERATORY LOGICZNE

! && ||

OPERATOR PRZYPISANIA

=

OPERACJA PRZYPISANIA WARTOŚCI

zmienna = wyrażenie;

```
int i, j, k;
```

deklaracje zmiennych

```
float x = 1.5;
```

```
float y = 1.5e-5;
```

inicjalizacja

```
char znak = 'A';
```

```
i = i + 3;
```

brak inicjalizacji

```
3 = x;
```

zła kolejność

```
x + y = x - y;
```

```
znak = 65;
```

błąd

Zmienne lokalne nie są domyślnie inicjowane - jeśli nie przypiszesz wartości to zmienna będzie posiadała pewną (losową) wartość. Dlatego warto zawsze inicjować wartości zmiennych.

	przypisanie	porównanie
C	$a = 3$	$a == 3$
Pascal	$a := 3$	$a = 3$
pseudo-kod	$a \leftarrow 3$	$a = 3$

*	mnożenie	$x * y$	$x \cdot y$
/	dzielenie	x / y	$\frac{x}{y}$
+	dodawanie	$x + y$	$x + y$
-	odejmowanie	$x - y$	$x - y$
%	reszta z dzielenia (modulo)	$x \% y$	$x \bmod y$

- Reszta z dzielenia (modulo, %) określona jest tylko dla argumentów całkowitych
- Operator dzielenia / dla argumentów całkowitych realizuje **dzielenie bez reszty!**

$1 / 2 \rightarrow 0$
 $1 / 2.0 \rightarrow 0.5$
 $1.0 / 2 \rightarrow 0.5$

PRIORYTETY OPERATORÓW

*	/	%	wyższy priorytet
+	-		niższy priorytet

Dla operatorów o takim samym priorytecie obliczenia są wykonywane od lewej do prawej.

$$z = \frac{x}{2y}$$

```
z = x / 2 * y;      /* złe */
z = x / (2 * y);
```

$$z = \frac{x - y}{x + y}$$

```
z = x - y / x + y;  /* złe */
z = (x - y) / (x + y);
```

W razie wątpliwości użyj nawiasów okrągłych.

```
#include<stdio.h>
```

stdio.h - **S**tandard **i**nput/**o**utput, biblioteka obsługująca komunikację ze standardowym wejściem i wyjściem aplikacji.

printf()

formatowane wyjście (wyświetlanie w terminalu)

```
printf("format", arg1, arg2, ... )
```

scanf()

formatowane wejście (odczyt z klawiatury)

```
scanf("format", adres1, adres2, ... )
```

FORMAT

specyfikator	znaczenie	przykład	wynik
%f	zmiennoprzecinkowa	<code>printf("%f",3.14);</code>	3.140000
%.2f	2 miejsca po przecinku	<code>printf("%.2f",3.14);</code>	3.14
%d	dziesiętna	<code>printf("%d",75);</code>	75
%c	znak	<code>printf("%c",75);</code>	K
%x	szesnastkowy	<code>printf("%x",75);</code>	4b

Przykład z większą liczbą argumentów

```
int a=2;
int b=3;
printf("Liczba %d plus %d wynosi %d\n", a, b , a + b);
```

Wynik:

Liczba 2 plus 3 wynosi 5

SYMBOLE SPECJALNE

symbol	znaczenie	przykład	wynik
\n	nowa linia	printf("n\nn")	n n
\t	tabulator	printf("t\tt")	t t
\"	"	printf("raz \"dwa\"")	raz "dwa"
\\	ukośnik \	printf("C:\\Users\\")	C:\Users\
%%	%	printf("%d%", 5)	5%
\?	?	printf("Jeszcze raz\?")	Jeszcze raz?

Przykład

```
printf("P\tj\tz%cr\te\ta\no\tst\tb\ng\tt\ta\nr\t ", 10);
printf("\tw\n%x\t \tn\nm\t \te\no\nw\na\nn\ni\nne\n", 10);
```

- specyfikator formatu powinien pasować do typu zmiennej

```
int a;  
scanf("%f", &a);
```

problem !!!

- drugi argument funkcji scanf to adres zmiennej &a
- formaty %f i %d pomijają początkowe białe znaki aż do napotkania wartości liczbowej
- znak niepasujący do formatu przerywa wczytywanie i pozostaje w strumieniu wejściowym

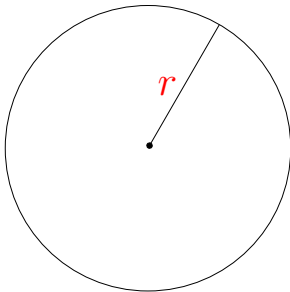
```
char a;  
float x,y;  
scanf("%f%f", &x, &y);  
scanf("%c", &a);
```

wczyta 2 liczby oddzielone białymi znakami

czyta pojedynczy znak

PRZYKŁAD: POLE I OBWÓD KOŁA

Problem: wyznacz pole i obwód koła o promieniu r .



$$P_{\circ} = \pi r^2 \quad O_{\circ} = 2\pi r$$

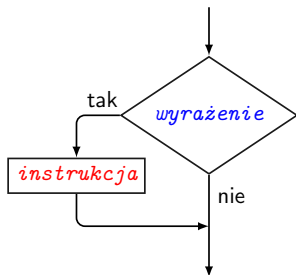
PRZYKŁAD: POLE I OBWÓD KOŁA

```
1  #include <stdio.h>
2  #define PI 3.14159
3
4  int main()
5  {
6      float r, pole, obw;
7
8      printf("Podaj promien kola\nr= ");
9      scanf("%f", &r);
10
11     pole = PI * r * r;
12     obw  = 2 * PI * r;
13
14     printf("Pole kola o promieniu %f wynosi %f\n", r, pole);
15     printf("Obwod kola o promieniu %f wynosi %f\n", r, obw);
16
17     return 0;
18 }
```

INSTRUKCJA WARUNKOWA IF ELSE

Składnia instrukcji if

```
if ( wyrażenie )  
    instrukcja
```



PRZYKŁAD:

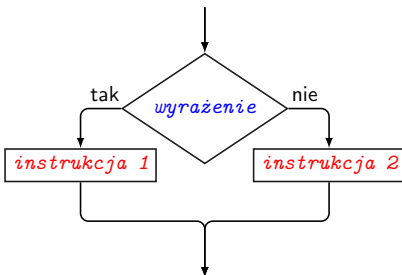
```
if( x > 0 ) printf("liczba dodatnia");
```

```
if ( x % 2 == 0 )  
{  
    printf("liczba parzysta");  
    x = x - 1;  
}
```

INSTRUKCJA WARUNKOWA IF ELSE

Składnia instrukcji if else

```
if ( wyrażenie )  
    instrukcja 1  
else  
    instrukcja 2
```



PRZYKŁAD:

```
if( x % 2 ) printf("liczba nieparzysta");  
else printf("liczba parzysta");
```

```
if ( x > 0 )  
{  
    printf("liczba dodatnia");  
    x = x - 1;  
}  
else x = 0;
```

operator	znaczenie	przykład	mat.
<	mniejszy niż	$x < y$	$x < y$
>	większy niż	$x > y$	$x > y$
<=	mniejszy lub równy	$x <= y$	$x \leq y$
>=	większy lub równy	$x >= y$	$x \geq y$
==	równy	$x == y$	$x = y$
!=	różny	$x != y$	$x \neq y$

operator	znaczenie	przykład	mat.
!	negacja (NOT)	!x	$\neg x$
&&	koniunkcja (AND)	x>1 && y<2	$x > 1 \wedge y < 2$
	alternatywa (OR)	x<1 y>2	$x < 1 \vee y > 2$

x	!x
0	1
1	0

x	y	x && y
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x y
0	0	0
0	1	1
1	0	1
1	1	1

```
if ( x > 0 )
    if ( x < 10 )
        printf("liczba wieksza od 0 i mniejsza niz 10");

if ( x > 0 && x < 10 )
    printf("liczba wieksza od 0 i mniejsza niz 10");

if (!(x > 0)) printf("liczba ujemna lub zero");

if ( !x > 0 ) printf("\?");

if ( x+1 > y-1 ) printf("\?");

if ( x || ! y && z ) printf("\?");
```

Kolejność operatorów: !, arytmetyczne, relacje, &&, ||
W razie wątpliwości użyj nawiasów okrągłych.

PRZYKŁAD: RÓWNANIE Z JEDNĄ NIEWIADOMĄ

Problem: znajdź miejsce zerowe funkcji liniowej

$$f(x) = ax + b$$

Algorithm Równanie z jedną niewiadomą

Dane wejściowe: współczynniki $a, b \in \mathbb{R}$

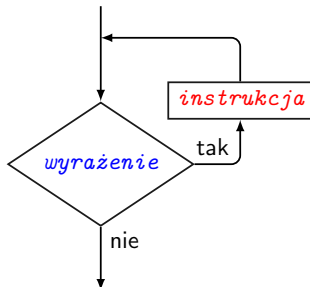
Wynik: miejsce zerowe $x_0 \in \mathbb{R}$ lub informacja o braku rozwiązania

- 1: **jeżeli** $a \neq 0$ **wykonaj**
 - 2: $x_0 \leftarrow -\frac{b}{a}$
 - 3: **wypisz:** x_0
 - 4: **w przeciwnym wypadku**
 - 5: **wypisz:** Brak rozwiązania
-

```
1  #include <stdio.h>
2
3  int main()
4  {
5      float a, b, x0;
6
7      printf("Podaj współczynniki równania\n");
8      printf("a = "); scanf("%f", &a);
9      printf("b = "); scanf("%f", &b);
10
11     if( a != 0.0 )
12     {
13         x0 = -b / a;
14         printf("x0 = %.4f\n", x0);
15     }
16     else printf("Brak rozwiązań\n");
17
18     return 0;
19 }
```

Składnia pętli while (dopóki)

```
while ( wyrażenie )  
    instrukcja
```



PRZYKŁAD

```
int n = 10;  
while( n > 0 )  
{  
    printf("%d\n", n);  
    n = n - 1;  
}
```

PĘTLA NIESKOŃCZONA

```
while(1) printf("C");
```

PRZYKŁAD: WYZNACZANIE SILNI $n!$

Problem: wyznaczenie wartości silni $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$

Algorithm Silnia

Dane wejściowe: liczba całkowita $n \geq 0$

Wynik: wartość $x = n!$

1: $i \leftarrow 2$

2: $x \leftarrow 1$

3: **dopóki** $i \leq n$ **wykonuj**

4: $x \leftarrow x \cdot i$

5: $i \leftarrow i + 1$

6: **wypisz** x

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int x, i, n;
6
7      printf("n = ");
8      scanf("%d", &n);
9
10     if(n<0) printf("Zle dane: n<0\n");
11     else
12     {
13         x=1;
14         i=2;
15         while( i <= n )
16         {
17             x = x * i;
18             i = i + 1;
19         }
20         printf("%d! = %d\n", n, x);
21     }
22     return 0;
23 }
```

TABLICA

przechowuje elementy tego samego typu, elementy identyfikowane liczbami (indeksem).

PRZYKŁAD

```
int tab[4];
```

Deklaracja tablicy 4 elementowej

```
tab[0] = 13;
```

```
tab[1] = 4;
```

```
tab[2] = -3;
```

```
tab[3] = tab[0] - 1;
```

```
tab[4] = -1;
```

Tablice w C są indeksowane od 0

Źle ! Poza zakresem.

	0	1	2	3
tab	13	4	-3	12

PRZYKŁAD: ODWRACANIE KOLEJNOŚCI

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int tab[100];
6      int i = 0;
7      int a = -1;
8
9      printf("Podaj sekwencje liczb calkowitych.\n");
10     printf("Aby zakonczyc podaj 0.\n");
11
12     while( a != 0 && i < 100)
13     {
14         scanf("%d", &a);
15         tab[i] = a;
16         i = i + 1;
17     }
18
19     printf("Podales %d liczb.\n", i);
20
21     while(i > 0)
22     {
23         i = i - 1;
24         printf("%d\n", tab[i]);
25     }
26
27     return 0;
28 }
```

```

1  #include<studio.h>;
2
3  char main()
4  {
5      int n
6
7      printf("Podaj liczbe calkowita wieksza od zera: ");
8      scanf("%f",&n);
9
10     if(n <= 0)
11         if ( n==0 ) printf("To jest zero!\n");
12     else
13     {
14         printf("Dzielniki liczby %d:\n",&n);
15         int i;
16         while( i<n );
17         {
18             if( n % i ) printf("%c/n",i);
19             i = i + 1;
20         }
21     }
22     return 0;
23 }
```

```

1  #include<stdio.h>                                /* studio.h, srednik */
2
3  int main()                                         /* int */
4  {
5      int n,i=1;                                    /* srednik , deklaracja , inicjalizacja */
6
7      printf("Podaj liczbe calkowita wieksza od zera : ");
8      scanf("%d",&n);                               /* format, adres */
9
10     if(n <= 0)
11     {                                              /* nawiasy */
12         if ( n==0 ) printf("To jest zero!\n");    /* == */
13     }
14     else
15     {
16         printf("Dzielniki liczby %d:\n",n);
17         while( i<=n )                             /* srednik , p. nieskonczona */
18         {
19             if( n % i == 0 ) printf("%d\n",i);    /* %d, \n, == */
20             i = i + 1;
21         }
22     }
23     return 0;
24 }

```

```
1 #include<stdio.h>
2 int nwd(int a,int b){int c;
3 while(b!=0){c=a%b;a=b;b=c;}
4 return a;} int main(){int a,
5 b;printf("Podaj dwie li"
6 "czby calkowite: ");scanf("%d %d",
7 &a,&b);printf("NWD(%d,%d) = %d\n",
8 a,b,nwd(a,b));return 0;}
```

 nwd-balagan.c

- Czytelność przede wszystkim
-  The International Obfuscated C Code Contest
- narzędzia dbające o czytelność
\$ astyle nwd-balagan.c

```
1  #include<stdio.h>
2  int nwd(int a,int b)
3  {
4  int c;
5  while (b!=0)
6  {
7  c=a%b;
8  a=b;
9  b=c;
10 }
11 return a;
12 }
13 int main()
14 {
15 int a,b;
16 printf("Podaj dwie liczby calkowite: ");
17 scanf("%d %d",&a,&b);
18 printf("NWD(%d,%d) = %d\n",a,b,nwd(a,b));
19 return 0;
20 }
```

 nwd-balagan2.c

STYL ALLMANA (BSD)

```
int main()
{
    int i;

    scanf("%d", &n);
    i=0;
    while (i < n)
    {
        if( i % 2 )
        {
            printf("%d\n", i);
        }
        i = i + 1;
    }
    return 0;
}
```

STYL K&R (GNU)

```
int main()
{
    int i, n = 100;

    scanf("%d", &n);
    i=0;
    while (i < n){
        if( i % 2 ){
            printf("%d\n", i);
        }
        i = i + 1;
    }
    return 0;
}
```

- Wewnętrzne bloki instrukcji wcięte względem zewnętrznych
- Instrukcje w jednym bloku zaczynają się w tej samej kolumnie
- Ogranicz długość linii do 80–120 znaków (szerokość ekranu)
- Rozbijaj długie instrukcje na kilka linii, zachowując czytelność:

```
while ((a < b && wplata(x) != -1) ||  
       (c > d && sprawdz(x) == 1)) {  
    if (w != NULL) {  
        a = a - 1 + sin(PI * 2);  
    }  
}
```

- Oddzielaj deklaracje zmiennych, funkcje i logiczne bloki kodu pustymi liniami
- Stosuj odstępy (spacje) między operatorami, po przecinkach
- język Python - wcięcia elementem składni języka

Komentarze pozwalają umieścić dodatkowe informacje dla czytających kod (nie są kompilowane)

```
/* komentarz blokowy */  
// komentarz liniowy wprowadzony w C99  
  
int main()  
{  
    /* Wszystko co tutaj jest napisane  
       jest komentarzem i nie zostanie skompilowane */  
  
    int x;    /* Bardzo wazna zmienna */  
    int y;    // Komentarz do konca linii  
}
```

Pisz **krótkie, klarowne komentarze** wyjaśniające "dlaczego", a nie "co" (kod powinien być samodokumentujący się).

Nie komentuj oczywistych rzeczy.

```
i = i + 1;    /* Zwiększenie licznika o 1 */  
i = i + k;    /* Ustawienie indeksu na ostatni element */
```

STANDARD C99

- funkcje `inline`
- deklaracje zmiennych w dowolnym miejscu w programie
- typ logiczny (`bool`), `long long int`
- tablice o zmiennej liczbie elementów
- komentarze w stylu C++ `//` to jest komentarz
- biblioteki, np.: `complex.h`, `stdbool.h`

- każdą zmienną trzeba zadeklarować (określić typ i nazwę)
- nie zapomnij o średniku na końcu instrukcji
- operator przypisania `a=b`, operator porównania `a==b`
- najpierw deklaracja zmiennej potem użycie (ANSI C)
- inicjuj zmienne wartościami początkowymi
- czytelność kodu bardzo ważna:
nieczytelny kod najprawdopodobniej jest błędny,
zrozumiałe nazwy zmiennych, wcięcia, komentarze
- spójny styl w całym projekcie

-  David Griffiths, Dawn Griffiths „*Rusz głową! C.*”, Helion, Gliwice, 2013.
-  „Kurs programowania w C”, WikiBooks,
<http://pl.wikibooks.org/wiki/C>
-  „C Programming Tutorial”, Tutorials Point,
<http://www.tutorialspoint.com/cprogramming/>