

# C言語再入門講座

# c言語再入門講座 目次

NEXT STEP

1. c言語とは
2. 変数と定数
3. 式と演算子
4. 文字と文字列
5. 配列
6. 多次元配列
7. 変数とポインタ
8. キャスト変換
9. 構造体
10. 共用体
11. 制御
12. 関数

# 1. C言語とは

- (1) プログラミング言語について
- (2) インタプリタとコンパイラ
- (3) C言語の歴史と位置付け
- (4) なぜ組み込み現場ではc言語が現役なのか
- (5) コンピューター内部での数字の扱いとn進数
- (6) 2進数、10進数、16進数の相互変換
- (7) 16進数での負の数の表し方

# 1. c言語とは

## (1) プログラミング言語について

プログラミング言語はコンピュータに解釈できるように作られた人工言語です。

コンピュータへの指令を行う為のプログラムを書くのに使われる言語です。

プログラミング言語は、人間がコンピュータに命令を指示するために作られており、コンピュータが曖昧さなく解析できるように設計されています。

多くの場合構文上の間違いは許されず、人間はプログラミング言語の文法に厳密にしたがった文を入力しなければなりません。

初期にはコンピュータが直接解読して動作できる機械語(マシン語)が用いられていましたが、

人の使う自然言語とあまりにもかけ離れていて、プログラムの作成効率が悪いため、人にとってわかりやすくしたアセンブラ言語が用いられるようになりました。

ただアセンブラ言語は基本的にマシン語と1対1に対応させただけなので、より人間に理解しやすい記号や代数表現を用いて書ける高級言語が開発されました。

プログラム言語の設計はその目的に応じてデータの形式、処理方法、文法などに違いが出てきます。

たとえば、計算機のシステムをつくるにはより機械語に近い処理が可能な Cが使われ、科学技術計算には数値の扱いに適したフォートランが使われます。

プログラミングの技法に応じて設計される場合も多く、手続き型言語のほか、論理構造を組み立てるのに適した PROLOGなどの論理型言語や関数を組み合わせて新しい関数をつくる LISPなどの関数型言語といった非手続き型言語があります。

また、C++などのオブジェクト指向プログラミング言語（→オブジェクト指向プログラミング）はプログラムの作成効率を上げるのに役立ちました。

機種依存性がない言語として Javaはインターネットに向いていることから世界中に広まりました。

近年、たくさんの新しいプログラミング言語が登場しています。

# 1. c言語とは

## (2) インタプリタとコンパイラ

どのような言語で書かれたプログラムもそのままでは実行できません。

マイコンが実行できるマシン語に直して初めてマイコンで実行できます。

高級言語で書かれたプログラムを一括してマイコンで実行可能なマシン語に変換する処理系をコンパイラ、ソースコードを1行ずつ解釈しながらマシン語に直して実行する処理系がインタプリタです。

主なコンパイラ系プログラム

C/C++、Fortran、Cobol

主なインタプリタ系プログラム

BASIC、Ruby、Python

コンパイラの特徴

インタプリタより実行速度が速い

コンパイルの手間がかかる

コンパイルした機械語のプログラムは他の環境（OSやCPUが異なる場合）では実行できない

※Java は基本的にはコンパイル方式の言語ですが、Java仮想マシンの機械語に翻訳され、仮想マシン上で実行されます。

コンパイル方式とインタープリタ方式の中間的な方式となります。

**スピードが要求される組み込みプログラムにはインタプリタは向いていません**

# 1. c言語とは

## (3) c言語の歴史と位置付け

C言語の歴史は古く、1972年頃UNIXの開発から生まれました。

C言語の 'C' の意味は？ と問われても明確に答えることは難しいと思います。

B言語の次に作成されたから「C言語」となり、それ以外'C'自体には特に意味がありません。

B言語の元とされたものはBCPL、その元となったのはCPLとなり、A言語は存在しません。

また、近年ではC言語に続くD言語の開発も進んでいるとのこと。

また、C++以外にもC#、PHP、Java、JavaScript、Objective-CなどC言語の影響を受けた言語も多数あります。

# 1. c言語とは

## (4) なぜ組み込み現場ではc言語が現役で使われ続けているのか

c言語は40年以上の歴史のある古い言語です、この40年間に新しい言語も多数でできました。  
ではなぜ組み込みの現場では今でもc言語が主流なんでしょうか。

### その①

コードが軽いので、資源が少ない環境や、制御などにリアルタイム性が要求される組み込みに最適な言語であるす。

### その②

開発資産や主流のソフトウェアがC言語でできています。

### その③

C言語は、アセンブラレベルと同等の処理を簡潔に記述でき、プログラマが意図したコードだけがコンパイラで生成されるため、マイコンの細かい制御が可能となります。この事はマイコンのレジスタを操作したりと組み込みプログラムには不可欠です。

# 1. c言語とは

## (5) コンピューター内部での数字の扱いとn進数

日常生活では基本10進法ですが、10進法以外では  
 時間が 秒未満は10進法、秒、分は60進法、時は12進法、または24進法 です。  
 また最近は余り聞かなくなりましたがダースという単位は12進法です。  
 コンピューターは0と1からなる 2進法で動いています。

なぜコンピューターは2進数で動いているのでしょうか？

電気信号のオンとオフでそのまま0と1を表すことができますが、これを10進数で実現しようとする、  
 例えば 0V→0、1V→1、2V→2 …… 8V→8、9V→9  
 のように電圧を読み込む(計る)回路や、正しい電圧を発生させる回路が必要となりとても複雑になってしまいますし、  
 とても現実的ではありません。

プログラムを書く時は数字を表すのに10進数や16進数を使います、コンパイラが10進数や16進数を自動で2進数になおしてくれます。

## (6) 2進数、10進数、16進数の相互変換


**2進数 → 10進数変換** 例：2進数 10011011 を 10進数に変換

128	64	32	16	8	4	2	1	
2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>	桁の重み
1	0	0	1	1	0	1	1	2進数
↓	↓	↓	↓	↓	↓	↓	↓	
128	0	0	16	8	0	2	1	← 加算
						=	155	10進数



# 1. c言語とは

**10進数 → 2進数変換** 例：10進数 155 を 2進数に変換

2)155	...	1	余りがある場合は 1	下位
2)77	...	1		
2)38	...	0	余りが無い場合は 0	
2)19	...	1		
2)9	...	1		
2)4	...	0		
2)2	...	0		
2)1	...	1		上位
0				

逆さになっている余りを横書きに直すと

**10011011**

155 (10進数) = 10011011 (2進数)

**2進数 → 16進数変換** 例：2進数 10011011 を 16進数に変換

128	64	32	16	8	4	2	1	
2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>	桁の重み
<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>1</b>	2進数
↓	↓	↓	↓	↓	↓	↓	↓	
8	0	0	1	8	0	2	1	← 加算
		=	<b>9</b>	<b>B</b>				16進数

# 1. c言語とは

## (7) 16進数での負の数の表し方

16進数の0xFFFFは符号有りとは符号無しでは10進数に直したときの値が異なります。

符号なし 0xFFFF → 65535

符号有り 0xFFFF → -1

符号有りの場合最上位ビットが1のときはマイナスの値を、最上位ビットが0のときはプラスの値を表します。

最上位ビット	符号の種類
1	マイナスの値を表す
0	プラスの値を表す

マイナス表現には**2の補数**を使用します

2の補数は、その対象の2進数を**全ビットを反転させ、+1**することで実現します。

2進数では2の補数を求める = マイナスの数ということもいえます。

2の歩数の求め方 -1の場合

0000 0000 0000 0001	1の2進数
1111 1111 1111 1110	0と1を反転する
1111 1111 1111 1111	1をたす
0xFFFF	16進数

## 2. 変数と定数

- (1) 変数とは
- (2) 変数の宣言
- (3) 変数の型とサイズと範囲
- (4) フォーマット指定子
- (5) Int型を使うときの注意点
- (6) 算術演算時の型
- (7) ビッグエンディアンとリトルエンディアン
- (8) 定数

## 2. 変数と定数

### (1) 変数とは

変数とはデータを格納しておく領域のことです。変数は通常メモリ上に確保され、値を代入したり参照したりすることができます。

### (2) 変数の宣言

変数は以下のように宣言します、宣言とは変数を使用するための定義です。

**記憶域クラス名 型修飾子 型名 変数名;**

記憶域クラス名	説明
auto	通常は省略
static	静的変数、関数の処理が終了しても変数を破棄しない
const	初期化以外で変更不可、主に定数として使う
volatile	コンパイラの最適化を抑制する型修飾子
register	CPUの中にあるレジスタを使う、通常のメモリよりアクセスが早い
型修飾子	説明
short	短
long	長
long long	長長
signed	符号付き
unsigned	符号なし
型名	説明
void	型なし
char	文字型
int	整数型
float	単精度実浮動小数点型
double	倍精度実浮動小数点型

## 2. 変数と定数

### (3) 変数の型とサイズと範囲

型	サイズ(byte)	範囲	備考
void	-	-	型なし
unsigned char	1	0~255	文字列は unsigned charの配列
signed char	1	-128~127	
unsigned short int	2	0~65535	
signed short int	2	-32768~32767	
unsigned int	2 or 4	0~4294967295	処理系によってサイズが変わるので注意が必要
signed int	2 or 4	-2147483648~2147483647	処理系によってサイズが変わるので注意が必要
unsigned long int	4	0~4294967295	
signed long int	4	-2147483648~2147483647	
unsigned long long int	8	0~18446744073709551615	
signed long long int	8	-9223372036854775808~ 9223372036854775807	

2進数、8進数、10進数、16進数相互変換ツール

<https://hoge hoge .tk/tool/number.html>

## 2. 変数と定数

### (4) フォーマット指定子

出力・入力フォーマット指定子

フォーマット指定子とは、のprintf()、fprintf()、sprintf()、scanf()、fscanf()、sscanf()などの関数で使用する、表示形式を指定するための記述子です。

指定子	対応する型	説明(出力の場合)	説明(入力の場合)	使用例
%c	char	1 文字を出力する	1 文字を入力する	"%c"
%s	char *	文字列を出力する	文字列を入力する	"%8s", "%-10s"
%d	int, short	整数を10進で出力する	整数を10進数として入力する	"%-2d", "%03d"
%u	unsigned int, unsigned short	符号なし整数を10進で出力する	符号なし整数を10進数として入力する	"%2u", "%02u"
%o	int, short, unsigned int, unsigned short	整数を8進で出力する	整数を8進数として入力する	"%06o", "%03o"
%x	int, short, unsigned int, unsigned short	整数を16進で出力する	整数を16進数として入力する	"%04x"
%ld	long int	32bit整数を10進で出力する		"%10ld"
%lu	unsigned long int	符号なし32bit整数を10進で出力する		"%10lu"
%lo	long int, unsigned long int	32bit整数を8進で出力する		"%12lo"
%lx	long int, unsigned long int	32bit整数を16進で出力する		"%08lx"
%lld	long long int	64bit整数を10進で出力する		"%-10lld"
%llu	unsigned long long int	符号なし64bit整数を10進で出力する		"%10llu"
%llo	long int, unsigned long int	64bit整数を8進で出力する		"%12llo"
%llx	long int, unsigned long int	64bit整数を16進で出力する		"%08llx"
%f	float	実数を出力する	実数を入力する	"%5.2f"
%e	float	実数を指数表示で出力する		"%5.3e"
%g	float	実数を最適な形式で出力する		"%g"
%lf	double	倍精度実数を出力する		"%8.3lf"

## 2. 変数と定数

### 表示桁数の指定

表示桁数は<全体の幅>.<小数点以下の幅>で指定する。<小数点以下の幅>は、文字列の場合には最大文字数の意味になる。

指定例	出力結果	備考
<code>printf("[%8.3f]", 123.45678);</code>	<code>[ 123.456]</code>	小数点含めて8桁、小数点以下3桁
<code>printf("[%15s]", "I am a boy.");</code>	<code>[ I am a boy.]</code>	文字列を15文字幅で表示
<code>printf("[%6s]", "I am a boy.");</code>	<code>[I am a]</code>	最大文字数8で文字列を表示
<code>printf("[%8.3e]", 1234.5678);</code>	<code>[1.234e+3]</code>	最大表示8桁、小数点以下3桁で指数形式で表示

### リーディングゼロ(ゼロ埋め)の指定

数値フィールドの場合に、ゼロ詰めを指定することができる。桁数の指定のまえにゼロを付加する。

指定例	出力結果
<code>printf("[%08.3f]", 123.45678);</code>	<code>[0123.456]</code>
<code>printf("[%05d]", 1);</code>	<code>[00001]</code>

### 符号の指定の指定

数値の表示は、デフォルトではプラス記号を出さないで、付けたいときは+を指定する。

指定例	出力結果
<code>printf("[%+5d]", 32);</code>	<code>[ +32]</code>
<code>printf("[%+5d]", -32);</code>	<code>[ -32]</code>
<code>printf("[%+8.3f]", 1.414);</code>	<code>[ +1.414]</code>

### 右詰・左詰の指定の指定

デフォルトでは右詰なので、左詰にしたいときは桁数指定の前にマイナスをつけなければならない。

指定例	出力結果
<code>printf("[%15s]", "I am a boy.");</code>	<code>[I am a boy. ]</code>
<code>printf("[%8.3f]", 123.45678);</code>	<code>[123.456 ]</code>
<code>printf("[%5d]", 1);</code>	<code>[1 ]</code>

## 2. 変数と定数

### (5) int型を使うときの注意点

**変数の型とサイズと範囲**表に書いていますがint型のサイズは使用するプロセッサや処理系によって2バイトと4バイトの場合があります。

例として Arduino UNO は2バイト Arduino DUO は4バイトです。  
両方ともに Arduino IDE で開発できますし、相互の移植もほとんど変更なしでできます。

ここで int 型に関して次のような注意が必要となります。

16ビットでは**負の数字**なのが32ビットでは**正の数**になってしまう。

例えば、16ビットの**0xFFFF**は10進数で表現すると **-1** ですが、これを 32ビットであつかうと **+65535** となります。

	符号有り	符号なし
	0xFFFF	0xFFFF
16ビット	-1	65535
32ビット	65535	65535

次ページのマイコンチップによる、変数型のサイズの違いの例 を参照



## 2. 変数と定数

型名	説明	Arduino Uno		Arduino Due	
		変数が占めるサイズ(sizeof())	取り得る値	変数が占めるサイズ(sizeof())	取り得る値
bool (C++言語) _Bool(C言語)	真偽値(trueとfalse)を格納する。	1	true/false(実際には他の値をとることも可能)	1	true/false(実際には他の値をとることも可能)
char	1バイトの値を格納する。文字を格納する。	1	-128～127	1	-128～127
unsigned char	1バイトの値を格納する。文字を格納する。	1	0～255	1	0～255
short int	整数を格納する。	2	-32768～32767	2	-32768～32767
unsigned short int	非負整数を格納する。	2	0～65535	2	0～65535
int	整数を格納する。	2	-32768～32767	4	-2147483648～2147483647
unsigned int	非負整数を格納する。	2	0～65535	4	0～4294967295
long int	整数を格納する。	4	-2147483648～2147483647	4	-2147483648～2147483647
unsigned long int	非負整数を格納する。	4	0～4294967295	4	0～4294967295
long long int	整数を格納する。	8	-9223372036854775808～ 9223372036854775807	8	-9223372036854775808～ 9223372036854775807
unsigned long long int	非負整数を格納する。	8	0～ 18446744073709551615	8	0～ 18446744073709551615
float	浮動小数点数を格納する。	4	-3.4028235e+38～ 3.4028235e+38	4	-3.4028235e+38～ 3.4028235e+38
double	浮動小数点数を格納する。	4	-3.4028235e+38～ 3.4028235e+38	8	-1.79769313486232e308～ 1.79769313486232e308
long double	浮動小数点数を格納する。	4	-3.4028235e+38～ 3.4028235e+38	8	-1.79769313486232e308～ 1.79769313486232e308
float _Complex	複素数型を格納する。	8	-	8	-
double _Complex	複素数型を格納する。	8	-	16	-
long double _Complex	複素数型を格納する。	8	-	16	-

## 2. 変数と定数

前頁の表から、int型の変数を使っている場合お互いにプログラムを移植した場合、変数のサイズの違いにより意図しない動きとなったりバグの原因となったりします。  
これを回避するために以下のように型を別名で定義して、それを使うことにより移植した場合にも不具合が生じないプログラムとなります。

なおこの別名は **stdint.h** に定義されています。

型名(幅指定整数型)	説明	stdint.h にでの定義
int8_t	8ビットの整数を格納する。	typedef signed char int8_t
uint8_t	8ビットの非負整数を格納する。	typedef unsigned char uint8_t
int16_t	16ビットの整数を格納する。	typedef signed int int16_t
uint16_t	16ビットの非負整数を格納する。	typedef unsigned int uint16_t
int32_t	32ビットの整数を格納する。	typedef signed long int int32_t
uint32_t	32ビットの非負整数を格納する。	typedef unsigned long int uint32_t
int64_t	64ビットの整数を格納する。	typedef signed long long int int64_t
uint64_t	64ビットの非負整数を格納する。	typedef unsigned long long int uint64_t

## 2. 変数と定数

### (6) 算術演算時の型(暗黙の型変換)

一般に異なる型の変数間の混合演算では、劣勢な方の型は優勢な方の型に変換されてから演算が行なわれると考えてよい。

(実際は少し違うが、結果としてこのように考えてよい)

型の優勢, 劣勢は次のようになっている。

劣勢 char型(1byte) < short int型(2byte) < long int型(4byte) < float型(4byte) < double型(8byte) 優勢

(同じバイト数の型の場合 劣勢 signed < unsigned 優勢 となっている)

混合演算	C言語での解釈	例
int型とdouble型の混合演算	int型をdouble型に変換してから演算	<pre>int a=3,c; double p=1.5,q; の時 q=a+p; →qは4.5になる。 c=a+p; →演算では4.5になるが、         代入時に小数点以下が失われてcは4になる。</pre>
int型とchar型の混合演算	char型をint型に変換してから演算	<pre>int a=3,c; char b=5; の時 c=a+b; →cは8になる。</pre>

## 2. 変数と定数

### (7) エンディアンとは?

2byte以上のデータ型(short int 以上)は複数byteにデータを配置します。

エンディアンとは簡単に言えば「データの並び順」です。バイトオーダーとかバイト順とも言います。

データの先頭byteを小さいアドレスに置く方式⇒ビッグエンディアン、

データの先頭byteを大きいアドレスに置く方式⇒リトルエンディアン です。

**long int型 2882400001 = 0xABCDEF01**

ビッグエンディアン	0xAB	0xCD	0xEF	0x01
	1010 1011	1100 1101	1110 1111	0000 0001
リトルエンディアン	0x01	0xEF	0xCD	0xAB
	0000 0001	1110 1111	1100 1101	1010 1011

Intelのマイクロプロセッサ 8086からPentiumシリーズ **リトルエンディアン**

Motorolaのマイクロプロセッサ **ビッグエンディアン**

MIPS や ARM、SHなど どちらにもなれる(**バイエンディアン**)

**エンディアンの変換関数 (endian.h のインクルードが必要)**

ホストバイトオーダーはホストマシンのエンディアン (CPU依存)、ネットワークバイトオーダーはビッグエンディアンのことを指します

uint32_t htonl(uint32_t hostlong)	32bitのホストバイトオーダーをネットワークバイトオーダーに変換する
uint16_t htons(uint16_t hostshort)	16bitのホストバイトオーダーをネットワークバイトオーダーに変換する
uint32_t ntohl(uint32_t netlong)	32bitのネットワークバイトオーダーをホストバイトオーダーに変換する
uint16_t ntohs(uint16_t netshort)	16bitのネットワークバイトオーダーをホストバイトオーダーに変換する

## 3. 式と演算子

- (1) 演算子の種類
- (2) 演算子の優先順位と結合規則
- (3) ビット演算子
- (4) 符号あり/なし
- (5) static変数
- (6) const
- (7) volatile
- (8) int型を使うときの注意点
- (9) 算術演算時の型
- (10) ビッグエンディアンとリトルエンディアン
- (11) 定数
- (12) 負の数の表し方(2の補数)

### 3. 式と演算子

演算子は演算の内容を指示する記号で、式は定数、変数、関数の返却値などを演算子を使って結合したものです。

算術演算子	構文	説明
+	$x + y$	xにyを加えます。
-	$x - y$	xからyを引きます
*	$x * y$	xにyを掛けます
/	$x / y$	xをyで割ります
%	$x \% y$	xをyで割った余りです

インクリメント・デクリメント演算子	構文	説明
++	++x	xに+1してxを評価します。(前置演算)
	x++	xを評価したあとxに+1します。(後置演算)
--	--x	xに-1してxを評価します。(前置演算)
	x--	xを評価したあとxに-1します。(後置演算)

ビット演算子	構文	説明
&	$x \& y$	論理積 (AND) を行います
	$x   y$	理和 (OR) を行います
^	$x \wedge y$	排他的論理和 (XOR) を行います
~	~x	否定 (NOT) を行います


シフト演算子	構文	説明
<<	$x << n$	xを左にnビットシフトさせます
>>	$x >> n$	xを右にnビットシフトさせます

※ sizeof も演算子として扱います

代入演算子	構文	説明
=	$x = y$	yをxに代入します
+=	$x += y$	xに ( $x + y$ )
-=	$x -= y$	xに ( $x - y$ ) を代入します
*=	$x *= y$	xに ( $x * y$ ) を代入します
/=	$x /= y$	xに ( $x / y$ ) を代入します
%=	$x \% = y$	xに ( $x \% y$ ) を代入します
&=	$x \& = y$	xに( $x \& y$ )を代入します
=	$x   = y$	xに( $x   y$ )を代入します
^=	$x \wedge = y$	xに( $x \wedge y$ )を代入します
<<=	$x << = y$	xに( $x << y$ )を代入します
>>=	$x >> = y$	xに( $x >> y$ )を代入します
比較演算子	構文	説明
==	$x == y$	xとyは等価である
!=	$x != y$	xとyは等価ではない
<	$x < y$	xはyより小さい(未満)
<=	$x < = y$	xはy以下である
>	$x > y$	xはyより大きい
>=	$x > = y$	xはy以上である
論理演算子	構文	説明
&&	$a \&\& b$	aとbが共に真の場合「真」
	$a    b$	aまたはbが真の場合「真」
!	!a	aが偽の場合「真」、aが真の場合「偽」

# 3. 式と演算子

演算子の優先順位と結合規則

種類線順位を	演算子	結合規則	優先順位
関数, 添字, 構造体メンバ参照,後置増分/減	() [] . -> ++ --	左→右	高
前置増分/減分, 単項式※	++ -- ! ~ + - * & sizeof	左←右	
キャスト	(型名)	左→右	
乗除余	* / %		
加減	+ -		
シフト	<< >>		
比較	< <= > >=		
等値	== !=		
ビットAND	&		
ビットXOR	^		
ビットOR			
論理AND	&&		
論理OR			
条件	?:		
代入	= += -= *= /= %= &= ^=  = <<= >>=	左←右	
コンマ	,	左→右	低

※ 式が複雑になる優先順位が分かりにくくなるので ( ) を使って優先順位り明確にする

# 演習用C言語実行環境

演習用にC言語の実行環境が必要となります。基本コンソールアプリを作成できる環境が必要です。

既にコンソールアプリ用の構築・実行環境がインストールされている場合は、インストールする必要はありません。

## 1. Visual studio 2019

Windows をお使いの場合 **Visual Studio 2019 communit**版をお勧めします。

ダウンロードはこちらから <https://visualstudio.microsoft.com/ja/downloads/>

Macをお使いの方も上記ページから **Visual Studio Community 2019 for Mac** をダウンロードできます。  
(ただし私がMacを持っていないので動作の確認はしていません)

## 1. Paiza.io

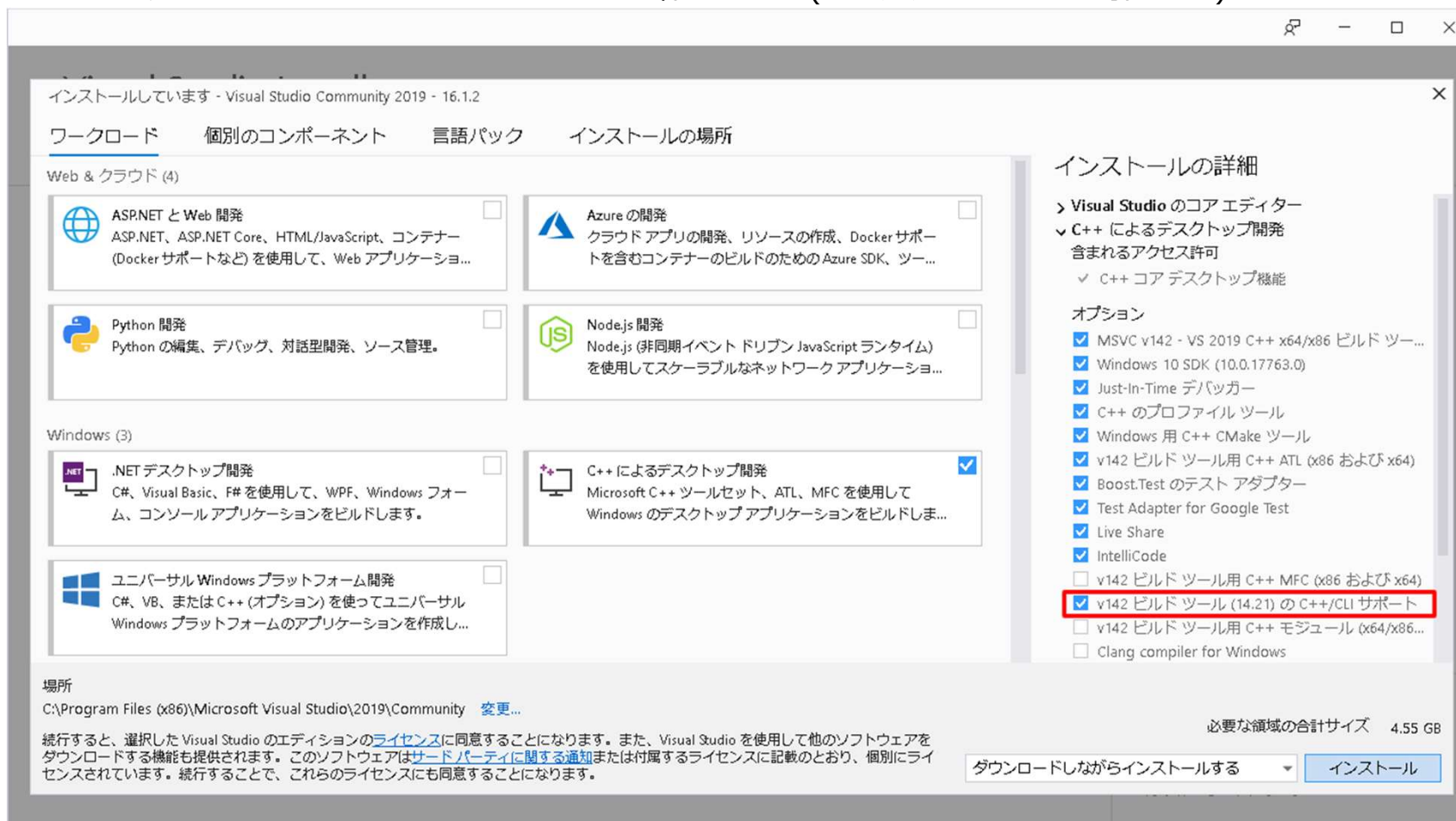
学習する為だけに Visual Studio 2019 をインストールするのに抵抗がお有りの方(結構ハードディスクの容量を必要とします)はブラウザ上で C言語のビルド・実行が出来る Paiza.io <https://paiza.io/> をお使いください。

演習問題のプログラムは Paiza.io でも問題なく実行できるように作成します。



## VS 2019インストールの際の注意点

VS 2019 のインストールの際、**CLIサポート** にチェックを入れてください。  
このチェックを入れないとコンソールアプリを作れません(後で追加することは可能です)。



## VS 2019での C言語プログラムの開発

VS 2019 C/C++ はデフォルトで C++プログラム用になっています。

そのままでC言語プログラムの開発も可能ですが、標準関数のヘッダファイルとかが違っていますので C言語の環境でのプログラム作成をお勧めします。

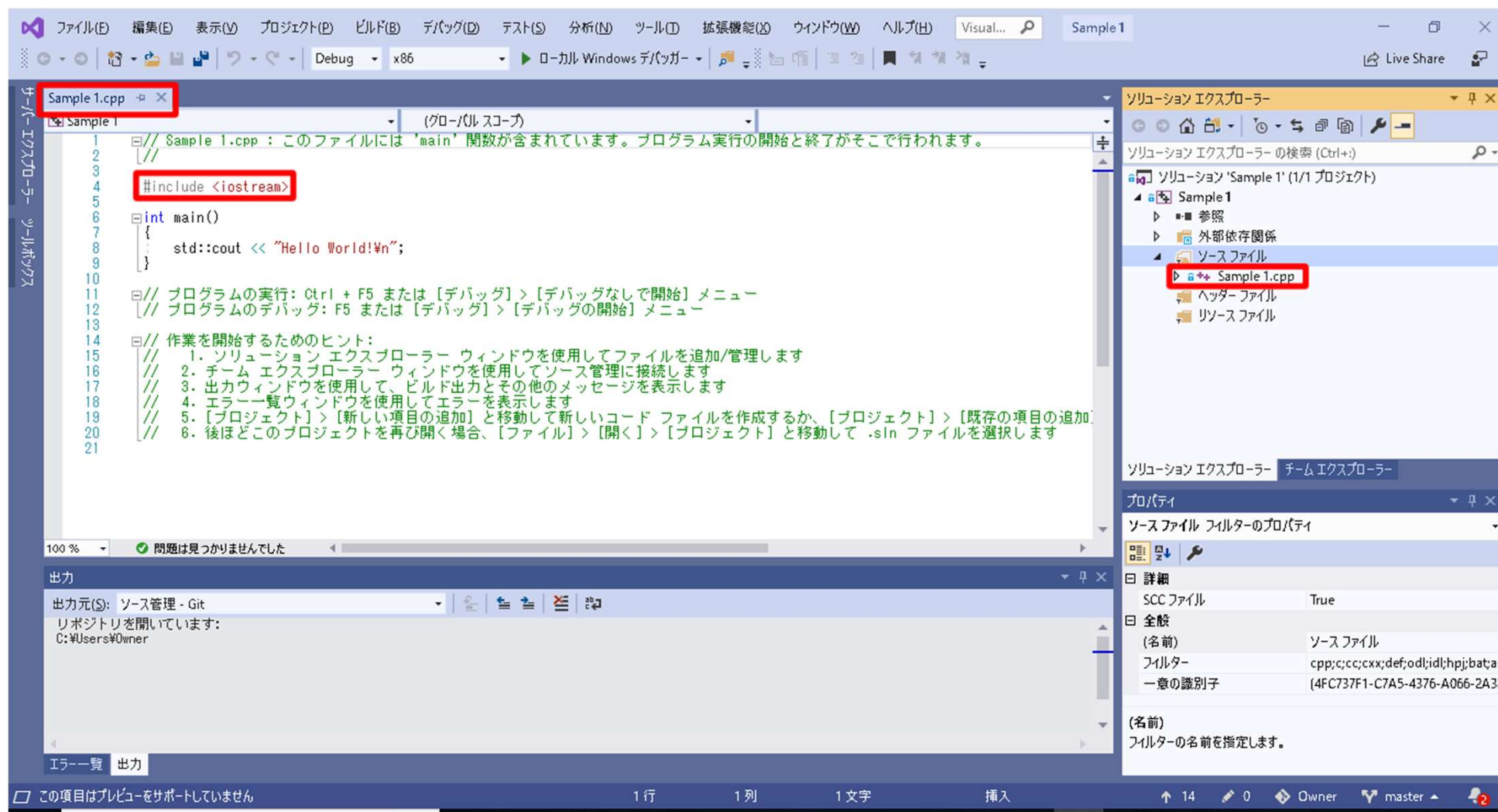
開発言語をC言語に設定する方法

新しいコンソールアプリプロジェクトを作成します。

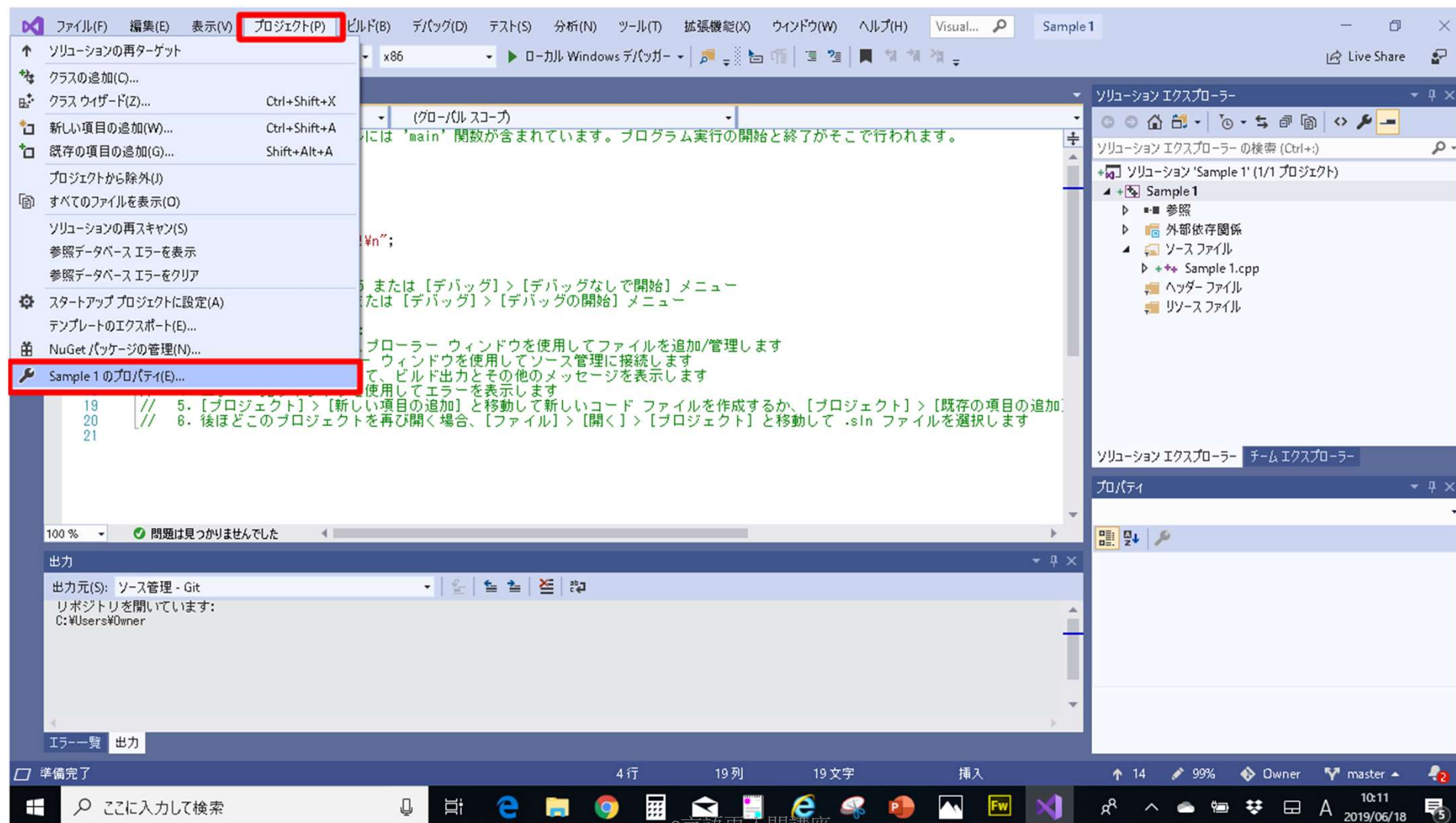
VS C/C++ の画面が次のようになって ソースファイルの拡張子が .cpp になっています。

以下のページで C言語 への設定方法を説明します。

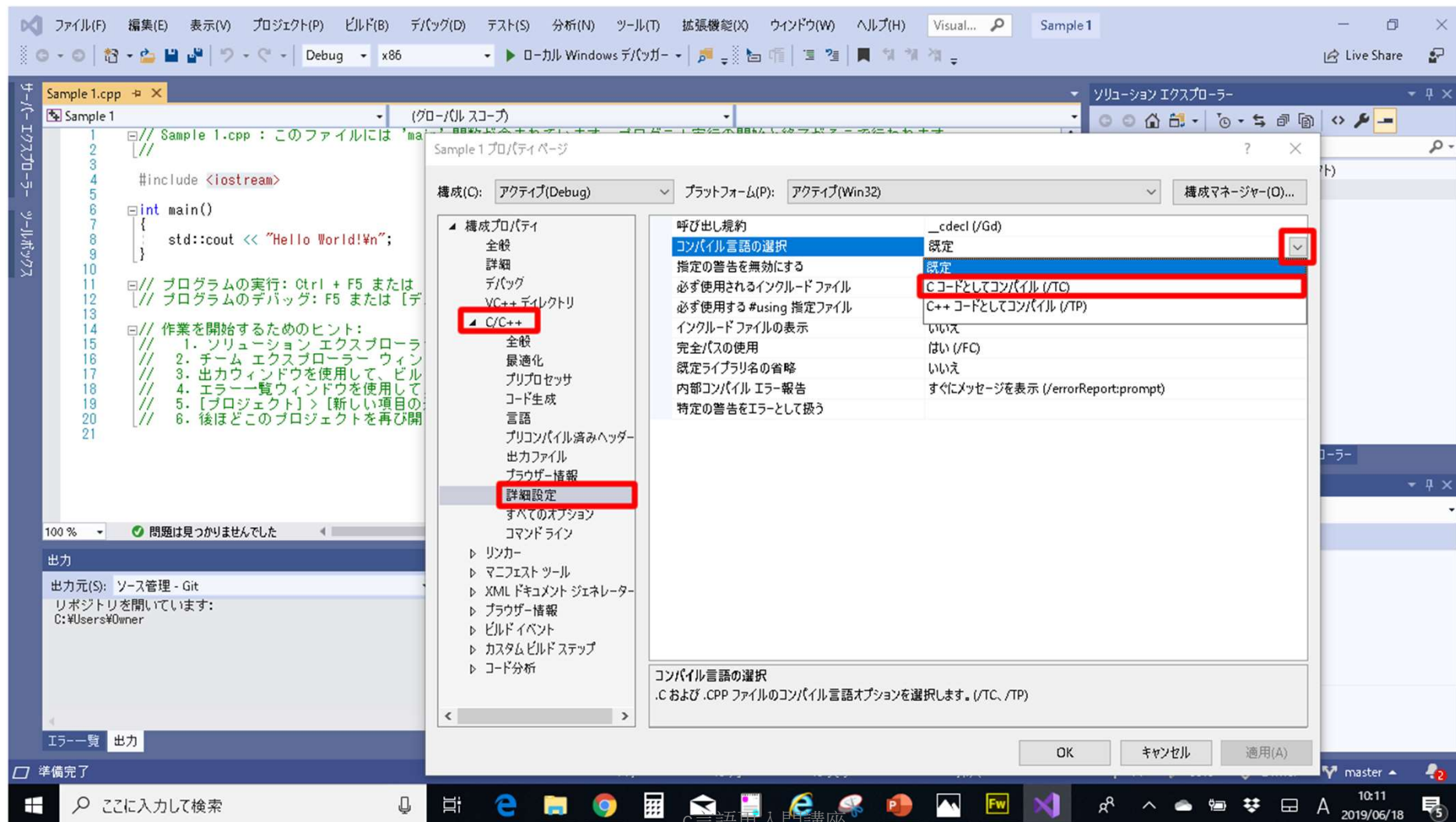
ソースファイル名が \*\*\*\*\*.cpp インクルードされているファイルが <iostream> となっています



[プロジェクト] [\*\*\*\*\*のプロパティ] をクリックします

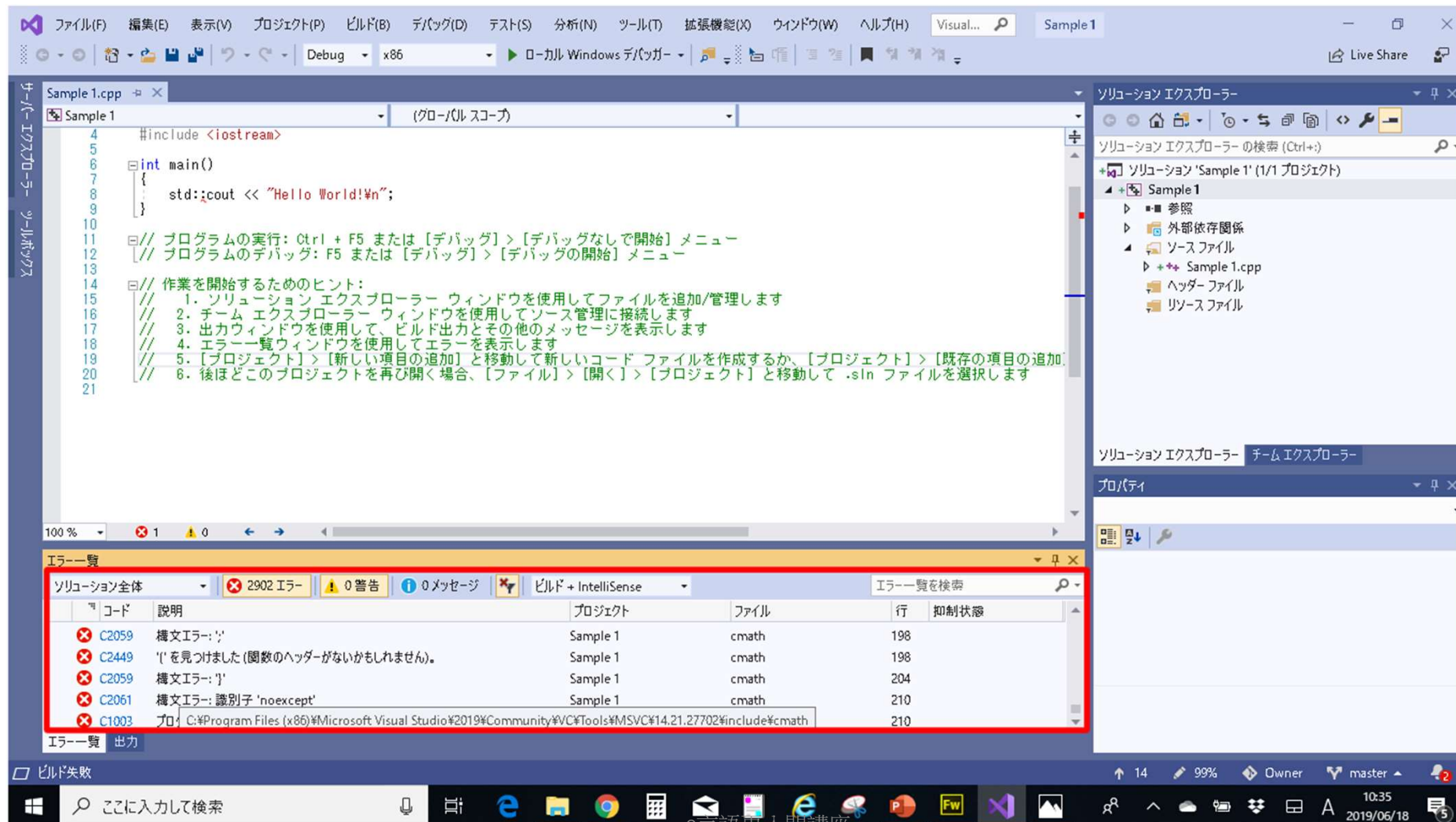


C/C++の詳細設定ページの コンパイル言語の選択 で Cコードとしてコンパイルを選択してください。





コンパイル言語をC言語に設定してビルドすると大量のエラーが発生します。



ソースファイル名を \*\*\*\*\*.c に変更して、プログラムもC言語で書き直してビルドするとエラーがなくなります

