

[戻る](#)

- [インライン関数](#)
 - [インライン関数の定義](#)
 - [何故処理速度が向上されるのか？](#)
- [デフォルト引数](#)
 - [引数を省略する](#)
 - [注意点](#)
- [関数のオーバーロード](#)
 - [オーバーロード？](#)
 - [オーバーロード使用時の注意点](#)
- [関数テンプレート](#)
 - [雛型を作る](#)
 - [利用する](#)
 - [注意点](#)

インライン関数

インライン関数の定義

処理が簡単な関数なら、インライン関数にすると処理速度が向上します

```
inline 戻り値の型 関数名(引数) { 関数内の処理 }
```

と定義します

何故処理速度が向上されるのか？

普通の関数だと、関数が定義されているソースに処理を参照するため
いくつも同じ関数を使用する場合、どうしてもその間の往復で時間が掛かります
インライン関数にすると、呼び出された部分に処理コードがコンパイラによって埋め込まれます
ソース間の往復が無いので、それだけ処理速度が向上されるわけです

デフォルト引数

引数を省略する

デフォルト引数で関数を定義すると、引数が必要な関数でも引数の指定を省略出来ます

```
戻り値の型名 関数名(型名 引数名 = デフォルト値);
```

と定義します

注意点

デフォルト引数を使用する場合、

```
右から順に設定しなければならない
```

と言う制約があります
つまり

```
int func(int a, int b, int c = 0, int d = 0);
```

とは定義出来ますが

```
int func(int a = 0, int b, int c = 0, int d);
```

のように、右側にデフォルト引数を設定していない引数がある関数は使えません

関数のオーバーロード

オーバーロード？

例えば、複数の数値の最大値を求める関数が必要だとします
普通ならint型の関数を作ればそれで終わりなのですが、
もし、小数の最大値を求める関数も必要になったらどうしましょう？
その場合、double型の関数も作らなければなりません
こうした「同じ処理でも、複数の関数を定義しなければならない」時に
C++では

```
引数の型や数が異なれば、同じ名前を持つ関数を複数定義できる
```

つまり

```
int max(int x, int y)
double max(double x, double y)
```

こうした定義が可能です
このように、引数の型や数が異なる同じ名前の関数を複数定義することを
「関数のオーバーロード」と言います

オーバーロード使用時の注意点

オーバーロードを使う際に注意すべき言葉あります

```
int func(int a);
void func(int a);
```

上のような、戻り値だけが異なる関数を複数定義しても、正常に作動しません
何故なら

```
func(5);
```

とあっても、どちらの関数を呼び出せばいいのか分からないからです
また

```
int func(int a, int b = 0);
int func(int a);
```

```
func(4);
```

のように、デフォルト引数を使った場合も、うまくオーバーロード出来ないことがあります

関数テンプレート

雛型を作る

上の場合だと、同じ処理なのに型が違うばかりに、関数を複数定義しなければなりません
そこで、関数のテンプレート（雛型）がC++では作れます

```
template <class テンプレート引数のリスト>  
関数の宣言（定義）
```

とテンプレートを宣言します

```
例  
template <class T>  
T maxt(T x, T y)  
{  
    . . .  
}
```

テンプレート引数（ここではT）を、仮の型名として定義します

利用する

上の例を使って説明します

```
int a, b;  
double da, db;
```

と、2種類（型）の変数があったとします
その時に

```
maxt(a, b);
```

と呼ばれた場合、Tにaやbのint型が代入されて処理されます
つまり

```
int maxt(int x, int y)  
{  
    . . .  
}
```

こうなります
同様にして

```
maxt(da, db);
```

と呼ばれた場合も

```
double maxt(double x, double y)  
{  
    . . .  
}
```

double型に変換されて処理されます

注意点

先のオーバーロードとテンプレートは似たものに見えますが、

使える局面が違います

関数のオーバーロード	関数内の処理が異なっても良い
関数テンプレート	型だけが異なって、同じ処理でなければならない