C++ 001 C++をはじめよう

【C 言語と C++言語】

C 言語と C++言語（以下 C++）は異なるプログラミング言語ですが、C++は C 言語を母体として作られているので、C 言語の知識、技術をそのまま使用できます。

C++は C 言語に＋α された言語なので、C 言語経験者にとって C++の学習は、その「＋α」の部分を学ぶことになります。

当テキストは C 言語を学習済みである事を前提にしています。

【C++コンパイラ】

C++のソースファイルの拡張子は「cpp」、コンパイルするには C++用のコンパイラが必要です。

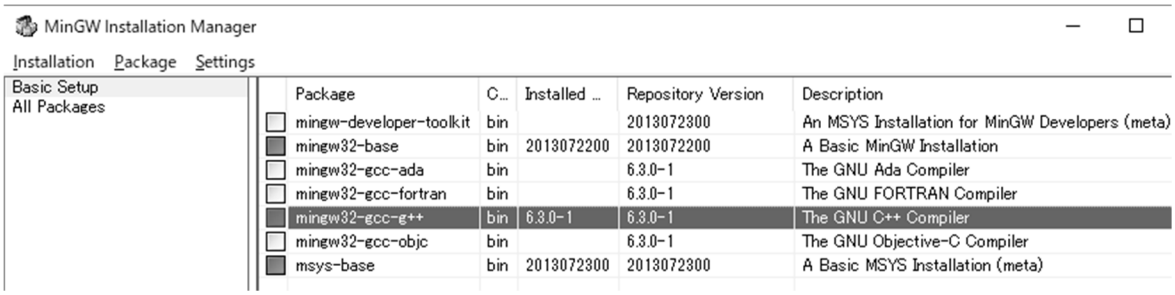
「C++ Compiler」「Visual C++」「MinGW（GCC）」は C 言語と C++と両方に対応したコンパイラです。

＜MinGW の場合＞

「mingw32-gcc-g++」のインストールが必要です。

MinGW のインストール時に「mingw32-gcc-g++」をインストールしていない場合は、下記の手順を行います。

1. 「MinGW Installation Manager」を起動する。
2. 左側のカテゴリの「Basic Setup」をクリックして
3. 「mingw32-gcc-g++」の左枠をクリックして「Mark for Installation」する
4. 「Installation」から「Apply Changes」をクリックしてインストールする。



コマンドは「gcc」を「g++」に変更します。

```
g++ -o *** ***.cpp
```

C++標準規格に準拠し、独自拡張のコードに警告を出す場合は「-pedantic」を付けます。

```
g++ -pedantic -o *** ***.cpp
```

C++で改良された点を紹介します。

【変数の宣言】

C 言語のローカル変数の宣言は関数ブロックの先頭（宣言部）に記述しますが、C++のローカル変数は関数ブロック内のどこでも宣言できます。

```
/* C 言語は宣言部で変数宣言を行う */
int main(void) {
    int var1;
    int var2;
    scanf("%d", &var1);
    scanf("%d", &var2);
```

```
//C++はどこでも変数宣言ができる
int main() {
    int var1;
    scanf("%d", &var1);
    int var2;
    scanf("%d", &var2);
```

C++のローカル変数のスコープは宣言したブロック内になります。

```
if (…) {
    int var;    //この変数 var はこのスコープ（ブロック）内で使用できる。
}               //処理がこのスコープを抜けた時、変数 var はメモリから破棄される。
```

```
for (int i = 0; i < 5; i++) {     //ここでも変数宣言が可能
    （略）
}

for (int i = 0; i < 10; i++) {    //スコープが異なれば同名で宣言できる。
    （略）
}
```

```
if (…) {
    int var;        //変数 var が宣言されている。
    for (…) {
        int var;    //同じスコープ内では同名で宣言できない。
    }
}
```

【const 変数】

C 言語では const 変数を、配列の要素数や case に使用できません。

```
const int VAR = 10;
char str[VAR];    /* const 変数は使用できない */
switch (…) {
    case VAR:      /* const 変数は使用できない */
```

C++では const 変数を、配列の要素数や case に使用できるように改良されました。

「const int VAR = 10;」と「#define VAR 10」は同じような使い方をしますが、const 変数はデータ型が指定できます。

【列挙体】

C 言語の列挙体は定数以外も代入できるので、選択性が低下してしまいます。

C++の列挙体は指定した定数以外は代入できないように改良されています。

```
enum collar { RED , GREEN , BLUE };

int main() {
    collar box;    //「enum」は省略できる。
    box = RED;     //「RED」「GREEN」「BLUE」以外は代入できない。
```

【引数の省略】

C 言語では関数の引数を省略する場合は「void」を指定します。

C++では関数の引数を省略する場合は void を省略して()のみで記述できます。

```
/* C 言語では */
void func(void) {
（略）
}

int main(void) {
    func();
```

```
//C++では
void func() {
（略）
}

int main() {
    func();
```

C++で追加された機能を紹介します。

【1 行コメント】

C 言語ではコメントを「/* コメント */」と記述しますが、C++はそれに加えて「//コメント」と記述できます。「/* コメント */」は複数行、「//コメント」は 1 行だけコメント化できます。

/* このコメントは 複数行をコメント化する。 */	//このコメントは 1 行だけコメント化する。

【bool 型】

bool 変数名;

C++で追加された新しいデータ型です。bool 型は「true」「false」の 2 つの値しか代入できません。C 言語では条件式の結果「真と偽」を「真=0 以外」「偽=0」と整数値で管理していますが、C++では「真=true」「偽=false」と bool 型で管理します。

【デフォルト引数】

デフォルト引数を指定すると、実引数を省略した場合にデフォルト値にすることができます。

＜デフォルト引数の定義＞

```
戻り値のデータ型 関数名(引数のデータ型 仮引数＝デフォルト値) {
```

＜例文 1＞

```
int main() {
    void func(int);         //関数プロトタイプ宣言
    func();                 //実引数を省略した場合は「10」になる。
    func(100);              //実引数を指定しているので「100」になる。
（略）
}

void func(int x = 10) {     //仮引数 x にデフォルト値「10」を指定。
```

デフォルト引数の記述は「関数定義」か「関数プロトタイプ宣言」か、どちらかに記述します。例文では関数定義に記述しました。

引数が複数ある場合、デフォルト値を指定した引数と、指定しない引数と混在させることは可能です。

＜例文 2＞

```
void func(int x = 10, int y = 20) {    //デフォルト引数を指定
（略）
}

int main(){
    func();             //実引数を省略。x は 10、y は 20 になる。
    func(100);          //第 2 引数は省略。x は 100、y は 20 になる。
    func(100, 200);     //実引数を指定。x は 100、y は 200 になる。
```

【関数のオーバーロード】

同じ関数名で、異なる引数の設定で定義することを「オーバーロード」と称します。
関数の呼び出し側は実引数に何を指定するかによって、呼び出される関数が異なる仕組みです。

```
void func() {（略）}                          //関数 1、仮引数無し
void func(int arg) {（略）}                  //関数 2、仮引数が int 型 1 個
void func(int arg1, int arg2) {（略）}       //関数 3、仮引数が int 型 2 個
void func(double arg) {（略）}               //関数 4、仮引数が double 型 1 個

int main() {
    func();          //実引数無し、関数 1 が呼び出される。
    func(10);        //実引数に int 型値 1 個、関数 2 が呼び出される。
    func(10, 20);    //実引数に int 型値 2 個、関数 3 が呼び出される。
    func(10.5);      //実引数に double 型値 1 個、関数 4 が呼び出される。
```

＜オーバーロードのルール＞

1. オーバーロードしたい関数と同じ関数名にして「引数の個数、データ型、順番」を変えること。
2. 「戻り値」は関係無い。

下記の関数をオーバーロードする場合

```
void func(int arg1 , double arg2 ) {}
```

下記のオーバーロードは可能

```
void func() {}                              //引数の個数が異なる。
void func(int arg ) {}                      //引数の個数が異なる。
void func(int arg1 , int arg2 ) {}          //第 2 引数のデータ型が異なる。
void func(double arg1 , int arg2 ) {}       //引数の順番が異なる。
```

下記のオーバーロードはエラーになる。

```
void func(int arg1 , double arg2 ) {}    //引数の個数、データ型、順番が同じ。
```

【インライン関数】

下記のソースコードは「引数付きマクロ定義」を関数のように使用しています。マクロは「マクロ名」を「記述したコード」に置換してからコンパイルされるため、通常の関数と比べて、処理速度が速くなるメリットとプログラムサイズが大きくなるデメリットがあります。

＜マクロを使用したコード＞

```
#define TRIANGLE(b, h) ¥
    double a = b * h / 2.0; ¥
    printf("三角形の面積は%.2lf です", a)

int main(void) {
    double b = 10;
    double h = 20;
    TRIANGLE(b, h);    /* マクロ「TRIANGLE」に引数が渡され置換される */
```

マクロの場合、引数や戻り値のデータ型のチェックが行われないのでエラーの要因に成りえます。
そこでC++では「インライン関数」を使用できます。

＜インライン関数の定義＞

```
inline 戻り値のデータ型 関数名 ( データ型 仮引数 ) {
```

関数定義の先頭に「inline」を付けるだけ。

＜例文＞

```
//インライン関数
inline void triangle(double b, double h) {
    double a = b * h / 2.0;
    printf("三角形の面積は%.2lf です", a);
}

int main() {
    void triangle(double, double);
    double b = 10;
    double h = 20;
    triangle(b, h);    //インライン関数を使用
```

先ほどの「マクロを使用したコード」と同様の処理を行っています。
インライン関数は、関数の呼び出し場所にソースコードを置換してからコンパイルされるため、マクロと同様の効果が得られます。