第７回C言語講座

１．　#define

#define は、さまざまな場面でプログラマにとって有効な手段です。#define の役割はコードの置き換えです。コンパイラはコンパイルを行う前に#define で定義されたコードを元のコードに置き換えてからコンパイルします。

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
C言語講座サンプル1
#define その1
by y.t
*/
#include <stdio.h> /*入出力のヘッダーファイル*/
#define ARRAYNUM        10
/*メイン関数*/
int main()
{
        int a[ARRAYNUM];
        int i;
        for (i = 0; i < ARRAYNUM; i++)
        {
                a[i] = i;
        }
        for (i = 0; i < ARRAYNUM; i++)
        {
                printf("a[%d] = %d¥n", i, a[i]);
        }
        return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

このプログラムではARRAYNUMというものを定義します。そして、その元々のコードは10です。コンパイラはコンパイルする前にすべての ARRAYNUM を 10 に置き換えてからコンパイルします。ではサンプル1のような#define の使い方にはどのような利点があるのでしょうか？ぜひ考えてみてください。

#define は置き換えです。このような使い方もできます。

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
C言語講座サンプル2
#define その2
by y.t
```

```
2006 第7回C言語講座（By y.t）
*/
#include <stdio.h> /*入出力のヘッダーファイル*/
#define RADIAN(x)        (x) * 3.14 / 180.0
/*メイン関数*/
int main()
{
        double a;
        scanf("%lf", &a);
        printf("a(RADIAN) = %lf¥n", RADIAN(a));
        return 0;
}
```

//

このような使い方をマクロ関数（マクロ）と呼んだりします。RADIAN マクロは度数法をラジアン値に直すマクロです。マクロは一見関数のように見えますが、あくまで置き換えです。

２．　ビット演算

ビットを使うと、さまざまな情報一まとめにできて大変便利です。ビット演算系の演算子を表 2.1 にまとめました。

表 2.1 ビット演算系の演算子

演算子	演算子名	機能
&	AND 演算子	ビットが互いに 1 のときのみ 1
|	OR 演算子	どちらかのビットが少なくとも 1 のときに 1
^	Ex-OR 演算子	互いにビットが異なるときに 1
~	NOT 演算子	ビットの反転
>>	右ビットシフト演算子	ビットを右にシフトします。
<<	左ビットシフト演算子	ビットを左にシフトします。
&=		AND 演算をして代入
|=		OR 演算をして代入
^=		Ex-OR 演算をして代入

実際にサンプルを見てみましょう。

//

```
/*
C言語講座サンプル3
ビット演算
by y.t
*/
#include <stdio.h> /*入出力のヘッダーファイル*/
/*メイン関数*/
int main()
```

```
{
	int a = 0x00FF0107;	//0000 0000 1111 1111 0000 0001 0000 0111
	int b = 0x0F071080;	//0000 1111 0000 0111 1010 0000 1000 0000
	printf("a & b = %X¥n", a & b);
	printf("a | b = %X¥n", a | b);
	printf("a ^ b = %X¥n", a ^ b);
	printf("~a = %X¥n", ~a);
	printf("a >> 8 = %X¥n", a >> 8);
	printf("b << 2 = %X¥n", b << 2);
	return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

各々の出力結果を2進数に変換し、各演算子の役割を実感してみてください。ちなみに、16進表記を用いているのは2進数に変換しやすくするためです。

３．　動的確保

配列を宣言するときその要素数は定数でした。これでは必要なときに必要な数だけの配列を使うことができません。

そこで、プログラムの実行中にメモリを確保し、必要な数だけ配列を確保してみましょう。

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
C言語講座サンプル4
動的確保
by y.t
*/
#include <stdio.h> /*入出力のヘッダーファイル*/
#include <stdlib.h>
#define ARRAYFUNC(NAME)	double NAME##(double *data, int size)
ARRAYFUNC(Sum)
{
	int i;
	double s = 0;
	for (i = 0; i < size; i++)
	{
		s += data[i];
	}
	return s;
}
ARRAYFUNC(Ave)
{
```

```
	return Sum(data, size) / size;
}
/*メイン関数*/
int main()
{
	double *data;
	FILE *fp;
	char filename[256];
	int datanum;
	int i;
	printf("Open filename...");
	scanf("%s", filename);
	fp = fopen(filename, "r");
	if (fp == NULL)
	{
		printf("NotFound...%s¥n", filename);
	}
	fscanf(fp, "%d", &datanum);
	data = (double*)malloc(sizeof(double) * datanum);	//datanum 個のメモリを確保
	for (i = 0; i < datanum; i++)
	{
		fscanf(fp, "%lf", &data[i]);
	}
	printf("Sum = %lf¥n", Sum(data, datanum));
	printf("Ave = %lf¥n", Ave(data, datanum));
	free(data);		//確保したメモリの開放
	return 0;
}
```

//

malloc 関数は引数で渡したサイズのメモリを確保してきます。これにより、配列のサイズを必要なだけ持ってくることができます。動的確保したメモリは free 関数で開放してやらなければなりません。これを忘れるとメモリが開放されないまま残ってしまい。空きメモリがなくなってしまいます。

４． 本日の演習

１． 0xFF00FF00 & 0x00FF00FF は？

２． 0x0848F011 ^ 0x000000FF は？

３． 動的確保を用いたほうが良いと思われるケースを考えてください。

５．　本日の関数

表5．1．本日の関数

関数	引数	戻り値	処理内容
void *malloc(size_t size);	size 確保するバイト数	確保した領域の先頭のポインタ	メモリの動的確保 必須ヘッダー stdlib.h malloc.h
void free(void *memblock);	memblock 開放するメモリブロック		確保した領域の開放 必須ヘッダー stdlib.h malloc.h