第 1 日[Advance]

データの扱いとデータ型

まず、C 言語におけるデータについて区別をします。C 言語においてデータは大きく「定数」と「変数」に区別されます。定数は値を変えないもの、変数は任意の値を取りうるものです。例えば、「a = 10;」であれば a が変数、10 が定数となります。

定数には、以下のような区別があります。

数値定数	整数定数	8 進数	先頭に「0」をつけて表記（077 など）
		10 進数	通常の 10 進数と同じ表記
		16 進数	先頭に「0x」をつけて表記（0x41 など）
	浮動小数点定数	小数点以下が扱える数（16.25 や 1.0e-3 など）	
文字定数	1 文字のこと。'A' や 'B' のように ' ' で囲んで表記		
文字列定数	複数文字のこと。"ABC" や "computer"のように " "で囲んで表記		

ここで、定数をどの型として扱うか、を区別するにはサフィックスを用います。サフィックスは、データの後につけます。例えば 8 進数の「77」を unsigned long 型で扱うには「077UL」と表記します。

さて、データ型ですがこれは以下のようなものが C 言語にはあります。なお、C++では long long int は仕様化されていませんが、コンパイラによって使えるようになっていることもあります。

型種類	型名	サフィックス	長さ	値域
	(signed) char	-	1Byte	-128~127
	unsigned char	-		0~255
整数型	(signed) short	-	2Byte	-32768~32767
	unsigned short	-		0~65535
	(signed) int	-	4Byte	-2147483648～2147483647
	(signed) long (int)	L		
	unsigned int	U	4Byte	0~4294967295
	unsigned long (int)	UL		
	(signed) long long (int)	LL	8Byte	-4611686018427387904 ~ 4611686018427387903
	unsigned long long (int)	ULL		0~9223372036854775807
実数型	float	F	4Byte	1.17549e-38～3.40282e+38
	double	-	8Byte	2.22507e-308～1.79769e+308
	long double	L	16Byte	3.3621e4932～1.18973e+4932
型なし	void	-	-	-

実際は、長さは仕様化されていません。基本的に整数は「short≦int≦long≦long long」、実数は「float≦double≦long」というようになっています。

また、型名は typedef で別名をつけることも出来ます。例えば「typedef int seisu;」を入れておくと、「seisu n;」という宣言で int 型の変数 n を宣言することができます。

変数宣言

変数宣言は、「修飾 型名 修飾 2　変数名修飾 3=初期値,修飾 2　変数名修飾 3=初期値,…」の形で行えます。修飾には、以下のようなものがあります。

auto	変数の寿命指定で、定義されたブロックが終了すると消滅することを示す。
static	変数の寿命指定で、プログラム全体の終了時まで変数を残すことをしめす。
const	変数を読み取り専用にする。初期値以外に変えようとするとコンパイルエラーに。
signed	省略。基本的に char 型以外では指定はいらない。
unsigned	省略。
register	変数の確保方法の指定で、これが指定されると出来る限りレジスタに変数を確保し、早くアクセスできるようにする。出来なかった場合通常と同じになる。
volatile	volatile が指定された変数はコンパイラが感知できない方法で値が変更されるかもしれないことを宣言する。複数のスレッド等が同じ変数を共有するような、高度な並列処理を行う場合に使用する。
struct	構造体であることを示す。詳しくは後日。
union	共用体であることを示す。詳しくは後日。
enum	列挙型であることを示す。詳しくは後日。

修飾 2 にはポインタ型である場合に*を入れます(詳しくは後日)。また、修飾 3 では配列変数であるときに[]を書きます。配列変数の[]は複数つけることも出来、その場合の変数名を多次元配列と呼びます。例えば「int n[3][5]」という宣言で n という二次元配列が作れます。また、配列変数の初期化では、int n[3]={0,1,2}のように{}を用いることが出来ます。二次元配列なら n[2][3]={{1,2,3},{4,5,6}}のような形で可能です。

変数名についてですが、以下のような指定があります。

- 名前は半角のアルファベット、数字、_(アンダーバー)のみで構成されます。
- 名前の先頭 1 文字には数字が使えません。
- 同一スコープ(ブロック)内に既に存在する名前は付けられません。
- 大文字と小文字は区別され、別のものとして扱われます。
- 予約語、関数名、型名などと重複することはできません。
- 名前の長さは 31 文字以内でなければいけません。

これを満たしていればプログラム上は問題ないのですが、やはり目的にあった分りやすい名前をつけるべきです。例えば数を数える変数なら「count」や「cnt」などです。

標準入出力関数について

代表的な標準入出力関数には、ほかに入力:「getchar() 」「gets() 」、出力:「putchar()」「puts()」がある。char がつく二つの関数は 1 文字を入出力し、s のつく二つの関数は一行を入出力する。具体的には、以下のような雰囲気のソースになる(使う際は stdio.h のインクルードが必要である点は注意)。

```
int main(){
        int a; /*int 型で宣言する/
        a=getchar() /*getchar は引数なし*/
        putchar(a) /*int 型変数の文字を表示*/
}
```

```
int main(){
        char a[256]; /*char 型で配列は十分に*/
        gets(a)
        puts(a)
}
```

なお、-s 型関数では、引数はアドレスである(つまり「&変数名」)。配列変数の場合、配列名なのだがこれには事情がある。詳しくは今後あつかう。ちなみに、puts()関数は最後に自動で\n を入れる。

実は、scanf()を含め、これらの入力関数は使用が推奨されないので、注意して使うこと。

printf 関数・scanf 関数を詳しく

まずは名前についてみてみましょう。これは「print・f」、「scan・f」という風にわかれ、print と scan はそのままですが、f は「format」、書式という意味があります。つまり書式設定ができる入出力関数ということです。

さてこの書式設定についてみていきましょう。これには、書式化文字列というものを使います。これは、以下のような形をとります。

%[フラグ][最小フィールド][．][精度][h｜l｜L]フォーマット指定子

まずフラグには+と-が入り、+が入ると数値に符号が表示され、-が入ると左詰表示になります。また、#も入り、これが入るとフォーマットが o,x,X いずれかの場合は、0 以外ならば必ず出力値の前に 0,0x,0X をつけ、e,E,f,g,G の場合は出力値に強制的に小数点を入れます。これらは重複可能です。

最小フィールドには表示する最小文字数を正 10 進数で指定し、それ以下のときは右詰になります。

精度はフォーマットによって意味が変わります。たとえば、文字列であれば出力する最大文字数の指定になり指定数以上の文字数を出力することはできず、あふれた文字は切り捨てられます。整数であれば最小桁数の指定になりますこの場合、最小桁数以上の桁でも切り捨てられることはありません。最小桁数以下の場合は、上位に 0 が付加されます浮動少数の e や f であれば小数点以下の表示桁数の指定になり g または G であれば最大有効桁数の指定（つまり全体）になります。

h|l|L は長さ修飾子で、hh だと char 型、h は short 型、l は long 型、ll は long long 型、L は long double 型になります。

そしてフォーマット指定子には、変数のデータをどう表示するかが入ります。

指定子	意味
d, i	10 進符号付き整数
u	10 進符号無し整数
o	8 進符号無し整数
x, X	16 進符号無し整数（ X は大文字で出力）
e, E	指数形式浮動小数点数（ E は大文字で出力）
f, F	小数形式浮動小数点数（ F は大文字で出力）
g, G	e または f 形式のうち適した方（ G は大文字で出力）
a, A	16 進浮動小数点（ A は大文字で出力）
c	文字
s	文字列
p	ポインタの値
n	整数変数に出力済み文字数を格納
%	'%'の出力

また、エスケープシーケンスというものもあります。これは\n のようにかわった働きをするもので、以下のようなものがあります。なお、これは「\ + 文字」で一文字の扱いになります。

記号	意味	記号	意味
\a	ベル文字（アラート）	\?	?を表示
\b	1文字分戻る	\'	シングルクォーテーション(')
\f	ページ送り（クリア）	\"	ダブルクォーテーション(")
\n	改行、復帰	\0	ヌル
\r	同じ行の先頭に戻る	\N	8 進定数（N は 8 進数の定数）
\t	水平タブ	\xN	16 進定数(N は 16 進数の定数）
\v	垂直タブ	\\	\を表示

マクロについて

次にみるのはマクロです。これは#define の活用法です。プリプロセッサは様々あるのですが、分割コンパイルをし出してから必要になるものがほとんどですので、マクロについて触れておきます。

#define は、代表的な使い方は 1 日目プログラム 2 のようにやはり変わらない値に名前をつけておく、という形になるのですが、#define 命令はコンパイル時に指定した語を変換してくれるという強さから、別の使い方も存在します。例えば、「#define PRINT_TEMP printf("temp = %d\n",temp)」というものを定義しておけば、プログラム中で PRINT_TEMP を書くだけで temp=(temp の値)を表示する機能を実現できます。「#define PRINTM(X) printf("%d\n",X)」を定義すれば、PRINTM(変数)で、変数の値を表示して自動改行してくれるという機能を実現できます。

また、決まりきった形の式・関数を表現するのにも使えます。例えば台形の面積を求めるのであれば「 #define GET_TRAPEZOID_AREA(A,B,H) (((A) + (B)) * (H) / 2) 」で GET_TRAPEZOID_AREA(3,2,5)のように値を指定することで台形の面積を得るような関数を得られます。

しかし、これらのような使い方はあまり推奨されていません。その理由については以下を参考に考えてみてください。

[ヒント]「#define GET_TRAPEZOID_AREA(A,B,H) (A + B) * H / 2」という一般的な定義でこれを実行したとき、「s = GET_TRAPEZOID_AREA(up,down,h + 3);」の値はどうなってしまうか、を考えてみてください。ついうっかりこのようなことをやってしまうのです。

定数について

C 言語で不変の値を扱う方法というのはいくつかあります。まずは#define で決めてしまうこと。これでその単語を定数として扱えます。

2つ目に「const」修飾子を使って変数を宣言すること。これで宣言することで、変数を初期化した値以外に変えられなくなります。そのため、例えば「const int a=3;a=5;」のようなプログラムではコンパイルエラーが出ます。これで a が定数として扱えます。const 修飾子は、関数において定数を扱うときに使われます。もらった値を変えないように const を宣言しておくのです(例：void func(const int *a))。これはポインタでよく出てくるので覚えて置いてください。

3 つ目に「enum」を使う方法。enum{文字列,文字列,文字列,…文字列,文字列}のように使うことで、それぞれの文字列に自動的に順に整数が与えられます。例えば RPG の状態を文字列で処理するのに使いやすいです。また、番号が飛ぶなどで番号を指定する必要があるときは「enum{…,文字列=数値,…」とすることで指定でき、次の文字列はその数値の次から割り振られます。enum は多くの文字列を定数として扱うのに向きますが、整数しか使えない点が欠点です。

ヘッダファイルの作り方

最後に、自作ヘッダファイルの作り方を示しておきます。自作でヘッダファイルを作ることで、ファイルを分割してコンパイルすることができます。これは簡単なことで、分割するファイルに含まれる関数の宣言部分、つまり int sum(int a, int b){(処理)};の int sum(int a, int b);の部分だけを抜き出したファイルを作り、これを～.h の形で置いておきます。これらの中身は同名で～.c という形にしておくだけでよいです。これを main 関数の含まれるファイルでインクルードしてやることで、分割したファイルをまとめてコンパイルすることができます。