Ｃ言語の基礎Ｉ

０．目次

１．はじめに

１．１　簡単な例

簡単な例（ソースプログラム）でＣ言語を紹介する。

［例１］出力

	プログラム	解説
1	/* << a001.c >> */	・注釈は/*と*/で囲み、２行以上にわたってもよい。 ・空行はプログラム中どこにでも入れることができ、プログラムを見やすくするために使われる。
2	#include <stdio.h>	・stdio.hという名前のファイルをプログラム中に取り込む。 ・このファイルには入出力関係の規則などが書かれており、プログラム中で使う命令がうまく動作するように設定する。 stdio.hのように「.h」をつけたファイルをヘッダファイルという。
3	int main() {	・Ｃのプログラムはいくつかの関数から構成され、実行はmainという名前の関数から始まる。 ・関数を構成する文（コンピュータに指示を与える基本単位で、必ずｾﾐｺﾛﾝで区切る）は{と}で囲む。この部分を関数の本体という。
4	printf("sample¥n");	・画面に文字列sampleと出力する。 ¥nは改行を意味する。printfも関数のひとつ。
5	}	

ソースプログラムの作成

```
/* << a001.c >> */
#include <stdio.h>
int main() {
  printf("sample¥n");
}
```

ソースプログラムのコンパイル

```
% cc a001.c
```

ソースプログラムの実行

```
% ./a.out
sample
```

［例２］計算処理

	プログラム	解説
1	/* << a002.c >> */	
2	#include <stdio.h>	
3	int main() {	
4	int a,b;	・プログラム実行中に値が変わるデータを変数という。変数には名前をつけることができ、それを変数名という。 ・プログラム中の変数は、すべて関数の本体の始めで宣言しておかなければならない。そのための文を宣言文という。 ・変数がとる値は決められており、整数型と浮動小数点型がある。整数型にはint型,short型,long型がある。 int型は、-32768～32767 または -2147483648～2147483647 short型は、-32768～32767 long型は、-2147483648～2147483647
5	a = 123;	・代入文（変数＝式）という。
6	b = 2 * a;	・＝は、右辺の計算結果を左辺の変数に代入する。
7	printf("%d %d¥n",a,b);	・%dは対応する変数の値を出力するための形式で変換仕様という。変換仕様%dは対応する整数型の変数と同じ桁数の欄がとられ、その中に値が出力される。
8	}	

ソースプログラムの作成

```
/* << a002.c >> */
#include <stdio.h>
int main() {
  int a,b;
  a = 123;
  b = 2 * a;
  printf("%d %d¥n",a,b);
}
```

ソースプログラムのコンパイル

```
% cc a002.c
```

ソースプログラムの実行

```
% ./a.out
123 246
```

［例３］入力

	プログラム	解説
1	/* << a003.c >> */	
2	#include <stdio.h>	
3	#define MAX 1000	・記号名または記号変数を定義する。 プログラム中のMAXという文字列を すべて1000に置き換える。 ・記号名は大文字を使う習慣がある。 この機能により、プログラムが分かり やすくなる。 また、プログラムの変更が容易にできる
4	int main() {	
5	int a;	
6	scanf("%d",&a);	・%dは対応する変数の値を入力するための 形式で変換仕様という。 変換仕様%dは、入力を整数とみなし、 対応する整数型の変数に値を代入する。 &aと書くことに注意。
7	a = MAX * a;	
8	printf("a=%8d ¥n",a);	・変換仕様%8dは8桁の欄をとり、 右寄せで変数の値を出力する。
9	}	

ソースプログラムの作成

```
/* << a003.c >> */
#include <stdio.h>
#define MAX 1000
int main() {
  int a;
  scanf("%d",&a);
  a = MAX * a;
  printf("a=%8d ¥n",a);
}
```

ソースプログラムのコンパイル

```
% cc a003.c
```

ソースプログラムの実行

```
% ./a.out
123
a=   123000
```

［例４］計算処理

	プログラム	解説
1	/* << a004.c >> */	
2	#include <stdio.h>	
3	int main() {	
4	float x,y;	・浮動小数点型には、float型とdouble型がある。float型は、有効数字がおよそ10進6桁ぐらいで10の-37乗から10の38乗の範囲の数を扱うことができる。 double型は、有効数字の桁数および範囲がfloat型よりも大きい。
5	scanf("%f",&x);	・%fは対応する変数の値を入力するための形式で変換仕様という。 変換仕様%fは、入力を10進浮動小数点数とみなし、対応する浮動小数点型の変数にその値を代入する。
6	y = x * x;	
7	printf("x=%f¥n",x);	・変換仕様%fは、浮動小数点数を10進数に変換し、小数点以下6桁で表すのに必要な欄をとり、そこに出力する。
8	printf("y=%10.3f¥n",y);	・変換仕様%10.3fは、浮動小数点数を10進数に変換し、10桁の欄をとり、小数点以下3桁で出力する。
9	}	

ソースプログラムの作成

```
/* << a004.c >> */
#include <stdio.h>
int main() {
  float x,y;
  scanf("%f",&x);
  y = x * x;
  printf("x=%f¥n",x);
  printf("y=%10.3f¥n",y);
}
```

ソースプログラムのコンパイル

```
% cc a004.c
```

ソースプログラムの実行

```
% ./a.out
1.2
x=1.200000
y=     1.440
```

１．２　Ｃ言語プログラムの翻訳と実行

ccコマンドでソースプログラムの翻訳とリンクまで行い、実行可能プログラムを作成する。手順は次のようになる。

（１）プリプロセス

プリプロセッサが起動され、#で始まる行が処理される。
#で始まる行はプリプロセッサへの指示である。
#で始まる行はソースプログラムのどこに現れてもよい。
#includeや#defineなどの文が展開される。

入力	ソースプログラム（x.c）
出力	標準出力(通常画面)
コマンド	**% cc -E ファイル名**

（２）コンパイル

コンパイラを起動する。

入力	ソースプログラム（x.c）
出力	アセンブラプログラム（x.s）
コマンド	**% cc -S ファイル名**

（３）アセンブル

アセンブラを起動する。

入力	ソースプログラム（x.c）
出力	オブジェクトプログラム（x.o）
コマンド	**% cc -c ファイル名**

（４）リンク

ローダを起動する。
作成されたオブジェクトプログラムと、Ｃ言語ライブラリを連結して実行可能プログラムを作成する。

入力	ソースプログラム（x.c）
出力	実行可能プログラム（a.out）
コマンド	**% cc ファイル名**

●ｄｅｆｉｎｅ文

define文で、文字列や式の置換が行える。

```
1  /* << dp111.c >> */
2  #include <stdio.h>
3  #define MAX 1000    /* 文字列の置換 */
4  #define INPUT(a,b)  scanf("%d%d",&a,&b) /* 式の置換 */
5  #define OUTPUT(a,b) printf("%d %d¥n",a,b) /* 式の置換 */
6  main() {
7    int a,b;
8    INPUT(a,b);
9    OUTPUT(a+b,MAX);
10 }
```

実行結果

```
% cc -E dp111.c
＜＜途中省略＞＞
main() {
  int a,b;
  scanf("%d%d",&a,&b);
  printf("%d %d¥n",a+b,1000);
}
% cc dp111.c
% ./a.out
12 345
357 1000
```

コンパイラのオプションから、「#define 名前 値」と同等のことができる。

%cc -D名前=値 ファイル名

```
1  /* << dp112.c >> */
2  #include <stdio.h>
3  main() {
4    int a=234;
5    printf("%d ¥n",a+MAX);
6  }
```

実行結果

```
% cc dp112.c
dp112.c: In function ‘main’:
dp112.c:5: error: ‘MAX’ undeclared (first use in this function)
dp112.c:5: error:(Each undeclared identifier is reported only once
dp112.c:5: error: for each function it appears in.)
% cc -DMAX=1000 dp112.c
% ./a.out
1234
```

●ｉｎｃｌｕｄｅ文

include文でヘッダファイルの取り込みを行う。
ヘッダファイルには定数や型の宣言などが記述され、複数のソースプログラムから参照される。ファイル名には拡張子として.hがつけられる。

<ファイル名>の場合：標準のディレクトリ(/usr/include)から探す。
"ファイル名"の場合：(1)ソースファイルのあるディレクトリ、
(2)-Iオプションで指定したディレクトリ、
(3)標準のディレクトリ
の順に探され初めて見つかったものが採用される。

```
1  /* << dp121.c >> */
2  #include <stdio.h>
3  #include "dp121.h"
4  main() {
5    int a;
6    scanf("%d",&a);
7    printf("a = %d  MAX = %d ¥n",a,MAX);
8  }
```

実行結果

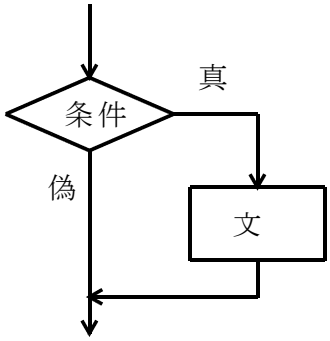
```
標準ディレクトリの確認。
% cd /usr/include
% ls -l stdio.h
-rw-r--r-- 1 root root 31568 11月  21 13:09 2012 stdio.h
ホームディレクトリに移動。
% cd
% ls
dp121.c
% cc dp121.c
dp121.hが見つからないというエラーメッセージがでる。
dp121.c:3:19: error: dp121.h: そのようなファイルやディレクトリはありません
dp121.c: In function ‘main’:
dp121.c:7: error: ‘MAX’ undeclared (first use in this function)
dp121.c:7: error: (Each undeclared identifier is reported only once
dp121.c:7: error: for each function it appears in.)
ヘッダファイル(dp121.h)の作成。
% cat > dp121.h
#define MAX 1000
^d
% cat dp121.h
#define MAX 1000
ヘッダファイル(dp121.h)の確認。
% ls
dp121.c     dp121.h
% cc dp121.c
% ./a.out
123
a = 123  MAX = 1000
```

２．条件分岐

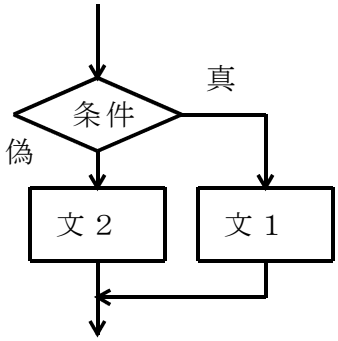
２．１　ｉｆ文

if文は一般につぎのように書く。

書き方	if(条件) 文
意味	条件が真の場合、文が実行される。

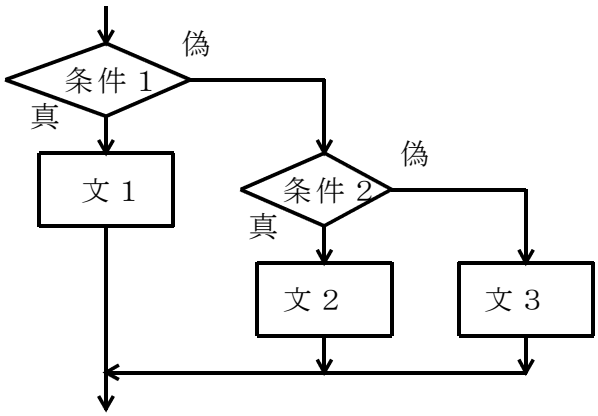


書き方	if(条件) 文1 else 文2
意味	条件が真ならば文1が、偽ならば文2が実行される。



書き方	if(条件1)
	　文1
	else if(条件2)
	　文2
	else
	　文3

意味	条件1が真ならば文1が実行され、条件1が偽でしかも条件2が真のとき、文2が実行される。 条件1も条件2も偽のとき、文3が実行される。



［例］1+2+…+10を求める。

```
1  #include <stdio.h>
2  int main() {
3    int i,s;
4    s = 0; i = 1;
5  loop:
6    if( i <= 10 ) {
7      s = s + i; i = i + 1;
8      goto loop;
9    }
10   printf("s=%d¥n",s);
11 }
```

（注意）文にはloopのような名札をつけることができる。
goto文で、名札（loop）のついた文へ飛び越す。

条件には値の大小を比較する関係式や論理式がよく用いられる。
たとえば、a==b, a*b<=c*d, (1<=a)&&(a<=n) などである。
==や<=を**関係演算子**といい、つぎの6種類ある。

a < b	aがbより小さい
a <= b	aがbより小さいか等しい
a == b	aがbと等しい
a != b	aがbと等しくない
a >= b	aがbより大きいか等しい
a > b	aがbより大きい

&&を**論理演算子**といい、つぎの3種類ある。

a && b	条件a,bがともに真のときに限り真になり、他の場合は偽になる。
a || b	条件a,bがともに偽のときに限り偽になり、他の場合は真になる。
!a	条件aが真のとき偽、偽のとき真になる。

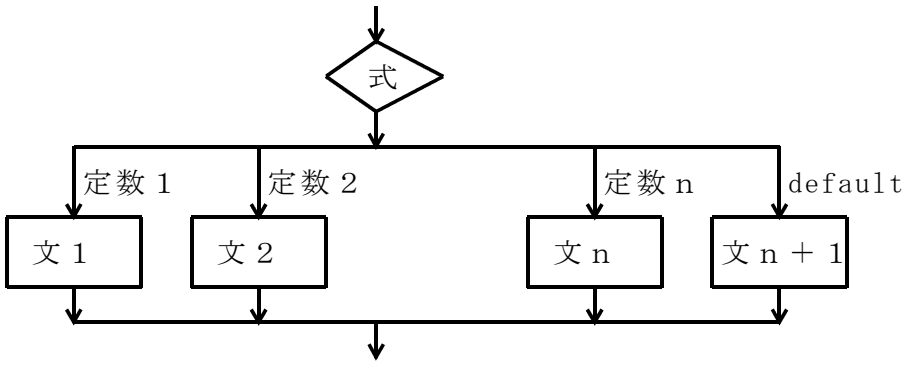
（注意）論理演算子は関係演算子よりも先に評価される。

２．２　ｓｗｉｔｃｈ文

switch文は一般につぎのように書く。分岐が多くなると便利である。

書き方	switch(式) {
	　case 定数1:
	　　文１；
	　　break;
	　case 定数2:
	　　文２；
	　　break;
	　　……
	　case 定数n:
	　　文ｎ；
	　　break;
	　default:
	　　文ｎ＋１；
	　　break;
	}

意味	式の値に応じて、定数i（1≦ｉ≦ｎ）のところへ分岐し 文ｉを実行する。 break文を実行すると、switch文は終了する。 式の値がどの定数iとも一致しないとき、defaultの ところへ分岐し、文ｎ＋１を実行する。



［例］1,2,3の数値を読み込み、それぞれの英単語を表示する。
　　1,2,3以外の数値の場合は、?を表示する。

```
 1  #include <stdio.h>
 2  int main() {
 3    int a;
 4    scanf("%d",&a);
 5    switch( a ) {
 6      case 1: printf("one ¥n"); break;
 7      case 2: printf("two ¥n"); break;
 8      case 3: printf("three ¥n"); break;
 9      default: printf("? ¥n"); break;
10    }
11  }
```

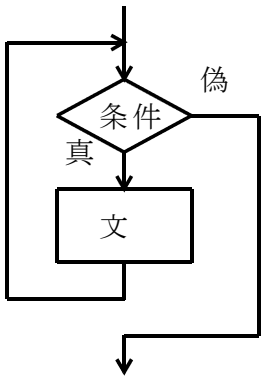
３．制御構造

３．１　ｗｈｉｌｅ文

while文は一般につぎのように書く。

書き方	while(条件) 文
意味	条件が真の間、文を実行し偽になるまで繰り返す。 最初から条件が偽の場合には、文は１回も実行されない。

このような構造を**whileループ**という。



［例］1+2+…+10を求める。

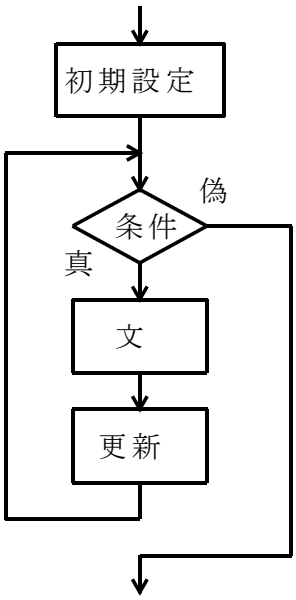
```
1  #include <stdio.h>
2  int main() {
3    int i,s;
4    s = 0; i = 1;
5    while( i <= 10 ) {
6      s = s + i; i = i + 1;
7    }
8    printf("s=%d¥n",s);
9  }
```

３．２　ｆｏｒ文

for文は一般につぎのように書く。

書き方	for(初期設定; 条件; 更新) 文
意味	まず初期設定が行われる。そして条件が真ならば文が実行され更新が実行される。その後条件が真の間、文と更新が実行され条件が偽になると文を実行せずにfor文のつぎの文を実行する。最初から条件が偽の場合、文は１回も実行されない。

このような構造を**forループ**という。



［例］1+2+…+10を求める。

```
1  #include <stdio.h>
2  int main() {
3    int i,s;
4    s = 0;
5    for( i=1; i<=10; i++ ) {
6      s = s + i;
7    }
8    printf("s=%d¥n",s);
9  }
```

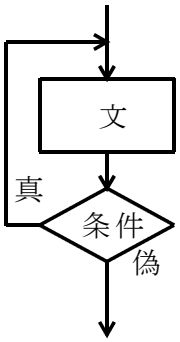
（注意）i++は、変数iに1を加えることを意味する。

３．３　ｄｏ文

do文は一般につぎのように書く。

書き方	do 文 while(条件);
意味	まず、文が１回実行される。 条件が真の間、文を実行し偽になるまで繰り返す。 文は少なくとも１回実行される。

このような構造を、**do-whileループ**という。



［例］1+2+…+10を求める。

```
 1  #include <stdio.h>
 2  int main() {
 3    int i,s;
 4    s = 0; i = 1;
 5    do {
 6      s = s + i; i = i + 1;
 7    }
 8    while( i<=10 );
 9    printf("s=%d¥n",s);
10  }
```

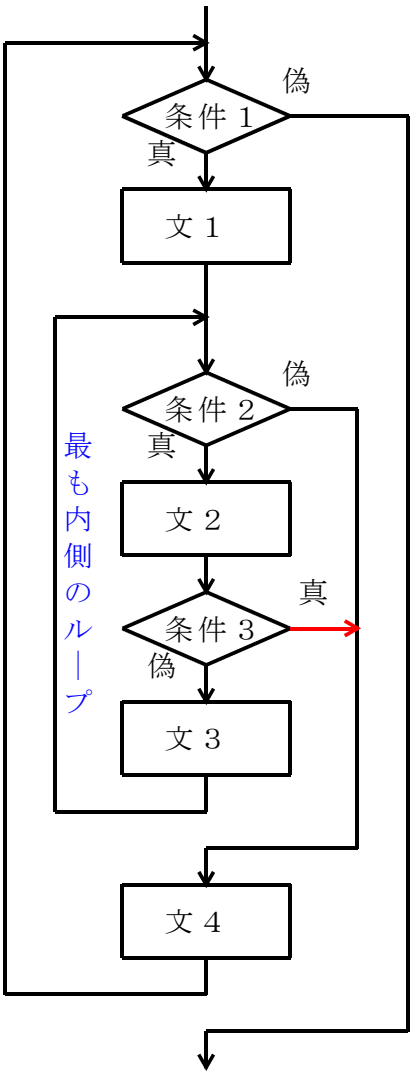
３．４　ｃｏｎｔｉｎｕｅ文，ｂｒｅａｋ文

whileやforなどのループから抜け出すときに、break文やcontinue文を使う。

break文を実行すると、このbreak文を含む最も内側のループを終了し、
つぎの文を実行する。

書き方	while(条件１) {
	　文１;
	　while(条件２) {
	　　文２;
	　　if(条件３) { break; }
	　　文３;
	　}
	　文４;
	}

意味	条件３が真になると、break文が実行され文４が実行される。



●break文の例

```
 1  /* << ba341a.c >> */
 2  #include <stdio.h>
 3  int main() {
 4    int i,j;
 5    for( i=1; i<=3; i++ ) {
 6      for( j=1; j<=3; j++ ) {
 7        printf("(1) i=%d j=%d¥n",i,j);
 8        if( i+j == 5 ) { printf("break¥n"); break; }
 9      }
10      printf("(2) i=%d j=%d¥n",i,j);
11    }
12    printf("(3) i=%d j=%d¥n",i,j);
13  }
```

実行結果

```
% cc ba341a.c
% a.out
(1) i=1 j=1
(1) i=1 j=2
(1) i=1 j=3
(2) i=1 j=4
(1) i=2 j=1
(1) i=2 j=2
(1) i=2 j=3
break
(2) i=2 j=3
(1) i=3 j=1
(1) i=3 j=2
break
(2) i=3 j=2
(3) i=4 j=2
```

●goto文で一気に複数のループを抜けることができる。

```
 1  /* << ba341b.c >> */
 2  #include <stdio.h>
 3  int main() {
 4    int i,j;
 5    for( i=1; i<=3; i++ ) {
 6      for( j=1; j<=3; j++ ) {
 7        printf("(1) i=%d j=%d¥n",i,j);
 8        if( i+j == 5 ) { printf("goto next¥n"); goto next; }
 9      }
10      printf("(2) i=%d j=%d¥n",i,j);
11    }
12  next:
13    printf("(3) i=%d j=%d¥n",i,j);
14  }
```

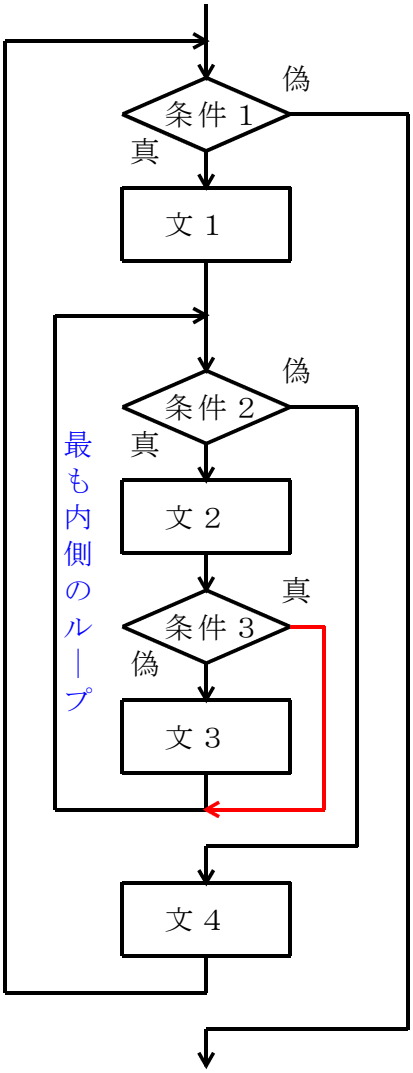
実行結果

```
% cc ba341b.c
% a.out
(1) i=1 j=1
(1) i=1 j=2
(1) i=1 j=3
(2) i=1 j=4
(1) i=2 j=1
(1) i=2 j=2
(1) i=2 j=3
goto next
(3) i=2 j=3
```

continue文が実行されると、このcontinue文を含む最も内側のループの残りの部分を無視して、条件の部分を調べる。

書き方	while(条件１) {
	　文１;
	　while(条件２) {
	　　文２;
	　　if(条件３) { continue; }
	　　文３;
	　}
	　文４;
	}

意味	条件３が真になると、continue文が実行され文３は実行されず条件２が調べられる。



●continue文の例

```
 1  /* << ba341c.c >> */
 2  #include <stdio.h>
 3  int main() {
 4    int i,j;
 5    for( i=1; i<=3; i++ ) {
 6      for( j=1; j<=3; j++ ) {
 7        printf("(1) i=%d j=%d¥n",i,j);
 8        if( i+j == 5 ) { printf("continue¥n"); continue; }
 9      }
10      printf("(2) i=%d j=%d¥n",i,j);
11    }
12    printf("(3) i=%d j=%d¥n",i,j);
13  }
```

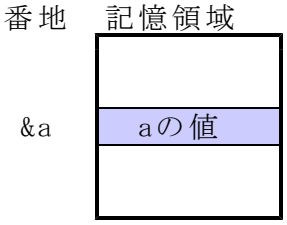
実行結果

```
% cc ba341c.c
% a.out
(1) i=1 j=1
(1) i=1 j=2
(1) i=1 j=3
(2) i=1 j=4
(1) i=2 j=1
(1) i=2 j=2
(1) i=2 j=3
continue
(2) i=2 j=4
(1) i=3 j=1
(1) i=3 j=2
continue
(1) i=3 j=3
(2) i=3 j=4
(3) i=4 j=4
```

４．変数、配列

４．１　変数

int a で変数aの宣言をする。
変数aの値を保存する記憶領域の位置（番地）が決定される。
変数aに対応づけられる番地を参照するには、&a とする。




```
 1  #include <stdio.h>
 2  int main() {
 3    int a;
 4    a = 123;
 5    printf(" a = %d ¥n",a);
 6    printf("&a = %x ¥n",&a); /* 16進数で出力。*/
 7    a = -a;
 8    printf(" a = %d ¥n",a);
 9    printf("&a = %x ¥n",&a);
10  }
```

実行結果

```
 a = 123
&a = effffb84
 a = -123
&a = effffb84
```

４．２　１次元配列

変数を多く使うとき、そのつど名前を考えなければならないとしたら困ってしまう。そこで、数学で使われる数列と同様の記法を考える。同じ型のデータを順に並べたものを配列という。配列はひとつの名前（配列名）をもつ。

各データを配列要素といい、配列名の後に添字を[と]で囲んで示す。添字は0から始まる。

配列要素a[i]の番地は&a[i]で参照する。

```
 1  /* << ba421.c >> */
 2  #include <stdio.h>
 3  int main() {
 4    int i;
 5    int a[9]; /* 配列名をaとし、9個の配列要素
 6                 a[0],a[1],...,a[8]を宣言する。*/
 7    a[0] = 1; a[1] = 1;
 8    for( i=2; i<=8; i++ ) {
 9      a[i] = a[i-1] + a[i-2];
10    }
11    for( i=0; i<=8; i++ ) {
12      printf("a[%2d] = %3d    ",i,a[i]);
13      printf("&a[%2d] = %x ¥n",i,&a[i]);
14    }
15  }
```

実行結果

```
% cc ba421.c
% ./a.out
a[ 0] =   1    &a[ 0] = e2a59eb0
a[ 1] =   1    &a[ 1] = e2a59eb4
a[ 2] =   2    &a[ 2] = e2a59eb8
a[ 3] =   3    &a[ 3] = e2a59ebc
a[ 4] =   5    &a[ 4] = e2a59ec0
a[ 5] =   8    &a[ 5] = e2a59ec4
a[ 6] =  13    &a[ 6] = e2a59ec8
a[ 7] =  21    &a[ 7] = e2a59ecc
a[ 8] =  34    &a[ 8] = e2a59ed0
実行ごとに割り当てられる番地は異なる。
% ./a.out
a[ 0] =   1    &a[ 0] = f3419ed0
a[ 1] =   1    &a[ 1] = f3419ed4
a[ 2] =   2    &a[ 2] = f3419ed8
a[ 3] =   3    &a[ 3] = f3419edc
a[ 4] =   5    &a[ 4] = f3419ee0
a[ 5] =   8    &a[ 5] = f3419ee4
a[ 6] =  13    &a[ 6] = f3419ee8
a[ 7] =  21    &a[ 7] = f3419eec
a[ 8] =  34    &a[ 8] = f3419ef0
```

（注意）int型は４バイト必要とすることから番地は４ずつ増加する。

４．３　２次元配列

int a[m][n]と宣言すると、

```
a[0][0],a[0][1],...,a[0][n-1]
a[1][0],a[1][1],...,a[1][n-1]
...
a[m-1][0],a[m-1][1],...,a[m-1][n-1]
```

のmn個の配列要素が決定される。

配列要素a[i][j]の番地は&a[i][j]で参照する。

```
1  /* << ba431.c >> */
2  #include <stdio.h>
3  int main() {
4    int a[3][3],i,j; /* 配列aの宣言。*/
5    a[0][0]=0;  a[0][1]=1;  a[0][2]=2;
6    a[1][0]=10; a[1][1]=11; a[1][2]=12;
7    a[2][0]=20; a[2][1]=21; a[2][2]=22;
8    for( i=0; i<=2; i++ ) {
9      for( j=0; j<=2; j++ ) {
10       printf("a[%d][%d]=%d    ",i,j,a[i][j]);
11       printf("&a[%d][%d]=%x ¥n",i,j,&a[i][j]);
12     }
13   }
14 }
```

実行結果

```
a[0][0]=0    &a[0][0]=effffb64
a[0][1]=1    &a[0][1]=effffb68
a[0][2]=2    &a[0][2]=effffb6c
a[1][0]=10    &a[1][0]=effffb70
a[1][1]=11    &a[1][1]=effffb74
a[1][2]=12    &a[1][2]=effffb78
a[2][0]=20    &a[2][0]=effffb7c
a[2][1]=21    &a[2][1]=effffb80
a[2][2]=22    &a[2][2]=effffb84
```

（注意）行優先で番地が割り当てられている。

４．４　局所変数の有効範囲

ブロック({}で囲まれた部分)内で宣言された変数（局所変数）はその内部でのみ有効である。すなわち、ブロック１内にブロック２が定義されているとき、

（１）ブロック１で宣言された変数は、ブロック２で同じ名前で宣言されない限り有効である。
（２）ブロック２で同じ名前で宣言されると、新たに宣言された方が優先する。

●（１）の例

```
 1  /* << ba441a.c >> */
 2  #include <stdio.h>
 3  int main() {
 4    int a;
 5    a = 111;
 6    printf("mainブロック内：a=%d¥n",a);
 7    {
 8      printf("  サブブロック内：a=%d¥n",a);
 9      a = 222;
10      printf("  サブブロック内：a=%d¥n",a);
11    }
12    printf("mainブロック内：a=%d¥n",a);
13  }
```

実行結果

```
mainブロック内：a=111
  サブブロック内：a=111
  サブブロック内：a=222
mainブロック内：a=222
```

●（２）の例

```
 1  /* << ba441b.c >> */
 2  #include <stdio.h>
 3  int main() {
 4    int b;
 5    b = 111;
 6    printf("mainブロック内：b=%d¥n",b);
 7    {
 8      int b;
 9      printf("  サブブロック内：b=%d¥n",b);
10      b = 222;
11      printf("  サブブロック内：b=%d¥n",b);
12    }
13    printf("mainブロック内：b=%d¥n",b);
14  }
```

実行結果

```
mainブロック内：b=111
  サブブロック内：b=0
  サブブロック内：b=222
mainブロック内：b=111
```