Cプログラミング1(再)
第9回

講義では、Cプログラミングの基本を学び
演習では、やや実践的なプログラミングを通して学ぶ

変数とメモリ

変数とメモリ

変数：「型」と「変数名」で宣言

例:　int i;

　　float a;

　　char c;

それぞれの型によって記憶できる値の範囲が異なる

変数は宣言しないと使えない

宣言：　変数の型と名前を決めること

表7.1　Cの型と表現できる範囲(抜粋)

符号なし整数

unsigned char	1バイト	0～255 (=2^8-1)
unsigned int	4バイト	0～4294967295 (= 2^{32}-1)

符号付き整数

char	1バイト	-128～127
int	4バイト	- 2147483647 ～ 2147483647 (=2^{31}-1)

実数

float	4バイト	有効桁約６桁(およそ10^{-38}～10^{38})
double	8バイト	有効桁約14桁(およそ10^{-308}～10^{308})

2種類の変数

自動変数（オート変数）

必要なときに用意され、不要になったら消される

何も宣言しなければこのタイプ

使う前に「初期化」が必要

静的変数（スタティック変数）

プログラムの開始時から終了時までメモリ上に確保

初期値が指定されていなければ、０に初期化

変数名について

変数名にはわかり易い名前をつける

よくある例：

c	ave
i, j, k	sum
n	p, q
x, y, z	ptr
len	count
str	nc
s, t	nl
max, min	nw
	buf
	tmp, temp

キーワード（予約語）に注意

Cのキーワードは

変数名にも関数名にも使えない

例: auto　double　int　break　if　else ...

註：現在のCコンパイラは、ほとんどが C++ (Cの上位言語）に対応しているため、C++ の予約語(例えば this ）も使えない！

ポインタ

ポインタの本来の意味：何かを指すもの

Cでは「メモリのアドレス」

　指し示されているものは

　　一つの変数、配列、構造体、関数のエントリーポイント

など

すでに我々にはscanfで馴染みの &変数は変数の「アドレス」を取り出す（変数へのポインタ）を返す

だから

scanf(“%d”, &x) では「変数xの(値を使うのではなく)アドレスに読み込んだ値を書き込む」ことができるのである

ポインタ(続)

ポインタを値とする変数の宣言方法(*を使う）：

例: int *ptr;

「整数」へのポインタを値とする変数ptr

これも馴染みである---ファイル操作で

FILE *fp;

としていたのは、変数fpの値はポインタ（アドレス）で、

それが指し示していたのはFILE情報の塊であった

特殊なポインタ NULL

NULL は0番地を指す、特殊なポインタ

ヌル

「指すものがない」ことを表すのに使われる

配列

配列は、ひとつの変数名によってたくさんのデータを扱うために用いられる

いわば変数が「一戸建ての家」（値を一つだけ記憶）

であるのに対し、配列は「アパート」（値を複数記憶、それぞれの部屋はインデックス（番号）で区別される

インデックスは０から始まることに注意

配列とメモリ

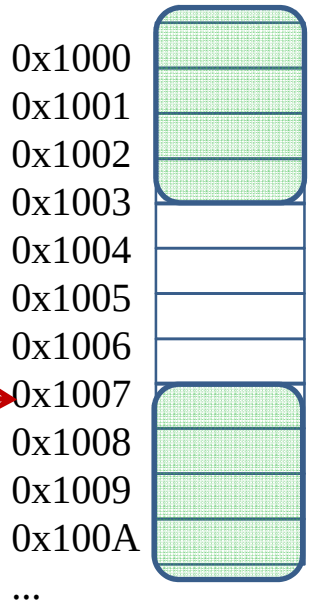
配列の変数名の値は「配列の先頭アドレス」

例: array[2]

array の値は配列の先頭アドレス（基準位置）

配列のインデックス(ここでは2)は、それからいくつ離れているか、を表す

だから、array[2]の値は *(array+2) と等価



一次元配列

int や float は4バイトの記憶領域をとる。それを  で表すと、

int array[10];

という変数宣言によって右のような
記憶領域が確保され、

array という変数には　0x1000

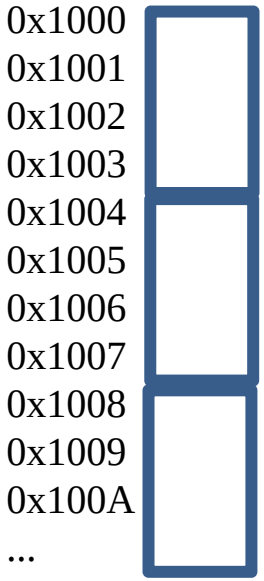
という「アドレス」が関連付けられる。

array[0] = *(array+0) は 0x1000 + 4*0 = 0x1000

array[1] = *(array+1) は 0x1000 + 4*1 = 0x1004

array[2] = *(array+2) は 0x1000 + 4*2 = 0x1008

と関連付けられる



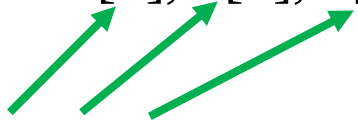
1次元配列の扱い

配列は「アパート」のようなもの

例:　int x[3];　　int だけが入居できるアパートx

インデックスは「号室」

x[0], x[1], x[2]という3室が作られる



- 値の初期化

  例:　int　x[3] = {1, 2, 3};　　// {} で要素をくくる

- 関数の引数(配列全体を扱う場合）

  関数呼出し：例　func(x);　// [ ] はつけない

  関数定義：例 int func(int ar[ ]) { … }　// [ ]をつける⇒配列の印

文字列 (=文字の一次元配列)

文字(char)は1バイトの記憶領域をとる。それを□で表すと、

char str[10];

という変数宣言によって右のような
記憶領域が確保され、

str という変数には　0x1000

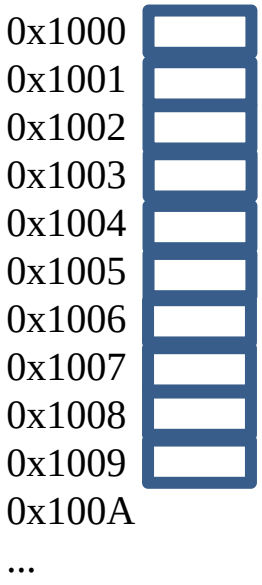
という「アドレス」が関連付けられる。

str[0] = *(str+0) は 0x1000 + 1*0 = 0x1000

str[1] = *(str+1) は 0x1000 + 1*1 = 0x1001

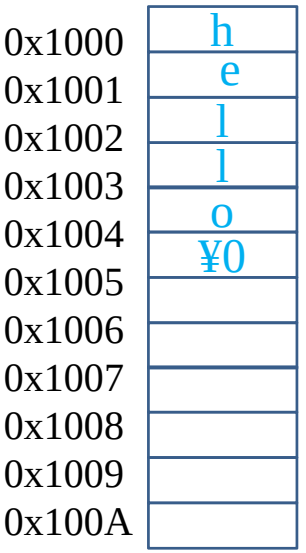
str[2] = *(str+2) は 0x1000 + 1*2 = 0x1002

と関連付けられる



文字データをメモリに格納

文字列 char a[ ]=”hello”;



文字列の最後には’¥0’(ヌル文字）が格納される

注意: ”hello” は
{ ’h’,’e’,’l’,’l’,’o’,’¥0’ }
と同じ意味

多次元配列

int や float は4バイトの記憶領域をとる。それを□で表すと、

int array[3][2];

という変数宣言によって右のような
(4*3*2=24バイト)記憶領域が確保され、
array という変数には　0x1000 という「アドレス」
が、またarray[1], array[2]にはそれぞれ以下のアドレスが
関連付けられる。

array[1] = 0x1000 + 4*2*1 = 0x1008

array[2] = 0x1000 + 4*2*2 = 0x1010

なお、

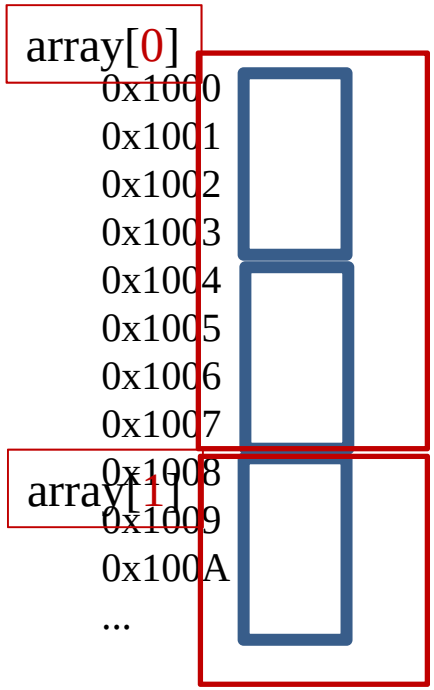
array[0][0] = *(array[0]+0) は 0x1000 + 4* (2 * 0 + 0) = 0x1000

array[0][1] = *(array[0]+1) は 0x1000 + 4* (2 * 0 + 1) = 0x1004

array[1][0] = *(array[1]+0) は 0x1000 + 4*(2 * 1 + 0) = 0x1008

…

と関連付けられる



多次元配列

変数宣言
int array[3][2]; によって
確保される記憶領域

	[0]	[1]
array[0]	array[0][0]	array[0][1]
array[1]	array[1][0]	array[1][1]
array[2]	array[2][0]	array[2][1]

多次元配列

int array[m][n];　(今の例では m=3, n=2)

の捉え方：
教科書(p.61)にあるように
「要素数nの配列がmこある」
とイメージする

アパートに例えると
　変数名と「最後以外の」インデックス
までがアパート名（例：ハイツarray1号棟)
　最後のインデックスが「号室」

だからこの場合、array[0], array[1]のような配列の要素は「ふつうの配列のように」使う

多次元配列の扱い

- 値の初期化

  例:　int　x[2][3] = {　{1, 2, 3}, {4, 5, 6}　};

x[0]に対する初期化

x[1]に対する初期化

x自体も配列なので {} でくくられる

- 関数の引数

  関数呼び出し　例　f1(x[0][1]);　　f2(x[1]);　　f3(x);

一個の整数 x[0][1]だけを 関数f1に渡す

整数配列 x[1]だけを 関数f2に渡す

多次元配列 x を 関数f2に渡す

  関数定義　例:

```
int f1(int a) { … }
```

```
int f2(int ar[ ]) { … }
```

```
int f3(int ar[ ][3]) { … }
```

文字列の配列

前述したように、

文字列　＝　文字の配列

それでは「文字列の配列」はどのようなものか？

今までの説明から、予想してみよう
(1)変数の宣言方法（書き方）
(2)文字列の配列の初期化の方法
(3)関数の呼出の方法
(4)文字列の配列を引数とする関数の定義方法