システムプログラミング序論
第2回
文字と文字列，多次元配列

大山恵弘

文字と文字列

文字型と文字コード

- 文字列とは文字の列
- 文字のデータ型が文字型
- 文字型の変数の宣言：

```
char c;
```

- 文字のデータは'で囲む

```
c = 'A';
```

文字の入出力

- printf の中では %c を使う

```
printf("Character is %c¥n", x);
```

- scanf でも同じ

```
scanf("%c", &x);
```

- 文字を扱うための関数も豊富にある

```
c = getchar(); /* 標準入力から文字を読み c に入れる */
putchar(c); /* c を文字として標準出力に出力する */
```

文字コード

- コンピュータの中では文字は数で表現されている
- 各文字を表現する数を，その文字の文字コードと言う
- 色々なコード体系がある
  - 英字，数字，記号を表す8ビット（1バイト）のコード
    - ASCII (American Standard Code for Information Interchange)
    - EBCDIC (Extended Binary Coded Decimal Interchange Code)
  - 日本語の文字（など）を表す複数バイトのコード
    - UTF-8
    - JIS (ISO-2022-JP)
    - Shift JIS
    - EUC-JP

ASCIIコード表

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01		SOH
^B	2	02		STX
^C	3	03		ETX
^D	4	04		EOT
^E	5	05		ENQ
^F	6	06		ACK
^G	7	07		BEL
^H	8	08		BS
^I	9	09		HT
^J	10	0A		LF
^K	11	0B		VT
^L	12	0C		FF
^M	13	0D		CR
^N	14	0E		SO
^O	15	0F		SI
^P	16	10		DLE
^Q	17	11		DC1
^R	18	12		DC2
^S	19	13		DC3
^T	20	14		DC4
^U	21	15		NAK
^V	22	16		SYN
^W	23	17		ETB
^X	24	18		CAN
^Y	25	19		EM
^Z	26	1A		SUB
^[	27	1B		ESC
^\	28	1C		FS
^]	29	1D		GS
^^	30	1E	▲	RS
^-	31	1F	▼	US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

https://msdn.microsoft.com/en-us/library/60ecse8t(v=vs.80).aspx
より抜粋

文字の比較

- 文字コードは数なので，比較できる
- `'A'` から `'Z'`，`'a'` から `'z'`，`'0'` から `'9'` は，文字コードでも順番に並んでいる
- 文字型の変数 `c` の値が大文字かどうかの検査：

```
'A' <= c && c <= 'Z'
```

- 数字かどうかを検査するには？
- 小文字かどうかを検査するには？

文字の変換

- 並んだ文字が文字コードでも順番に並んでいることを再び利用
- 小文字から大文字への変換：

```
c = c - 'a' + 'A';
```

（c = c - 97 + 65; と同じ）

- 数字から数への変換：

```
i = c - '0';
```

（i = c - 48; と同じ）

文字列とは

- 文字列は文字の列
- 文字列は文字の配列で表現される

```
char b[10];
```

- " で囲んだデータは文字列定数として扱われる

```
"world"
```

- printf や scanf では %s によって文字列を指定

```
printf("String is %s¥n", b);
scanf("%s", b);
```

b に & が付かないことに注意

文字列の終端

- 文字列とは「コード0で終わる文字の列」
  - コード0（'¥0'と書く）は，文字列の終端を示すコード
- サイズ10の文字型の配列を文字列 "ABC" で初期化すると，最初の4要素に 'A'，'B'，'C'，'¥0' が入る
- printf で %s を使うと，'¥0' の直前までの文字を出力する
- 文字列を格納するための配列は，文字列の長さ+1以上のサイズである必要があることに注意
  - "ABC" を格納するための配列のサイズは，少なくとも4

文字列の初期化

- 文字列定数による文字型配列の初期化を，配列の宣言時に書ける

```
char b[10] = "ABC";
```

- 後から代入はできない

```
char b[10];
b = "ABC";
```

```
arrayinit.c: In function 'main':
arrayinit.c:7:5: error: incompatible types when
assigning to type 'char[10]' from type 'char *'
   b = "ABC";
     ^
```

文字列の使用例

- 小文字を大文字に直して出力するプログラム

```
int main(void)
{
  int i;
  char s[10];
  printf("Please input string? ");
  scanf("%s", s);
  for (i = 0; s[i] != '¥0'; i++) {
    if (s[i] >= 'a' && s[i] <= 'z') {
      s[i] = s[i] - 'a' + 'A';
    }
  }
  printf("Result = %s¥n", s);
  return 0;
}
```

文字列のコピー

- 文字列は，「最後の文字が '¥0' である」ことを覚えておく

```
for (i = 0; s1[i] != '¥0'; i++) {
  s2[i] = s1[i];
}
s2[i] = '¥0';
```

関数に配列を渡す

- 関数の定義

```
関数が返す値のデータ型 関数名(配列のデータ型 配列名[], …)
{
    …
}
```

- 関数の呼び出し

```
関数名(配列名, …)
```

配列を引数に受け取る関数

- 配列の要素の和を求める

```
int sum(int a[], int n)
{
  int i, s;
  s = 0;
  for (i = 0; i < n; i++) {
    s += a[i];
  }
  return s;
}
```

文字列をコピーする関数

- 文字列を文字の配列として扱う

```
void string_copy(char s2[], char s1[])
{
  int i;
  for (i = 0; s1[i] != '¥0'; i++) {
    s2[i] = s1[i];
  }
  s2[i] = '¥0';
}
```

文字列を扱うライブラリ関数の例

- 文字列をコピーする関数 `strcpy(d, s)`
  - `d` に `s` をコピー
- 文字列を連結する関数 `strcat(d, s)`
  - `d` の末尾に `s` を連結
- 文字列を辞書順で比較する関数 `strcmp(s1, s2)`
  - `s1` が `s2` より辞書で先に出てくるなら−1，後に出てくるなら1を返す
  - `s1` と `s2` が同じ文字列なら0を返す

ライブラリ関数による文字列のコピー

- 文字の配列に文字列を代入することはできない
  - 文字列は代入するものではなくコピーするもの
- コピーには **strcpy** 関数（など）を使う

```
char p[10] = "ABC";
char q[10];
…
strcpy(q, p);
printf("p = %s¥n", p);
printf("q = %s¥n", q);
```

多次元配列

配列の使用例： 数列

```
int main(void)
{
  int a[11];
  int i;
  a[0] = 1;
  a[1] = 1;
  for (i = 2; i <= 10; i++) {
    a[i] = a[i - 2] + a[i - 1];
  }
  printf("a[10] is %d.¥n", a[10]);
  return 0;
}
```

配列の使用例：集合

```
int main(void)
{
  int a[10], i, flag;
  for (i = 0; i < 10; i++) {
    scanf("%d", &a[i]);
  }
  flag = 0;
  for (i = 0; i < 10; i++) {
    if (a[i] == 3) {
      flag = 1;
      break;
    }
  }
  if (flag) {
    printf("Number 3 was found.¥n");
  } else {
    printf("Number 3 was not found.¥n");
  }
  return 0;
}
```

配列の使用例： 表

- 10行×20列の表を考える
- どういう変数で表現する？
- 一つの案：
  - 縦×横のサイズを持つ配列を宣言

    ```
    int T[10 * 20];
    ```

  - 表の要素の参照と更新では，その要素が配列のどこに格納されているかを計算

    ```
    x = T[i * 20 + j];
    ```

  - 記述が面倒だし，誤りが入りやすい
  - そこで…

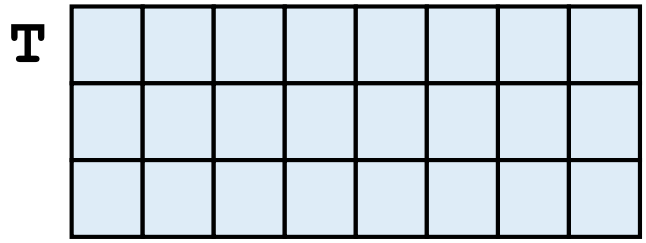
2次元配列

- 表のような，2次元データの処理をわかりやすく書くための仕掛け
- 宣言：

```
int T[3][8];
```

- 参照：

```
x = T[i][j];
```



- 要素の参照や更新において，行や列のサイズを気にしたり，配列の添字を計算する必要がなくなる

多次元配列

- 配列は3次元以上でも良い

```
int S[10][20][3];

double X[100][100][100][100];
```

文字列の配列

- 文字列は文字の1次元配列
- ということは，文字列の配列は2次元配列
- 宣言：

```
char 文字列の配列の変数名[文字列の最大数][文字列の最大長];
```

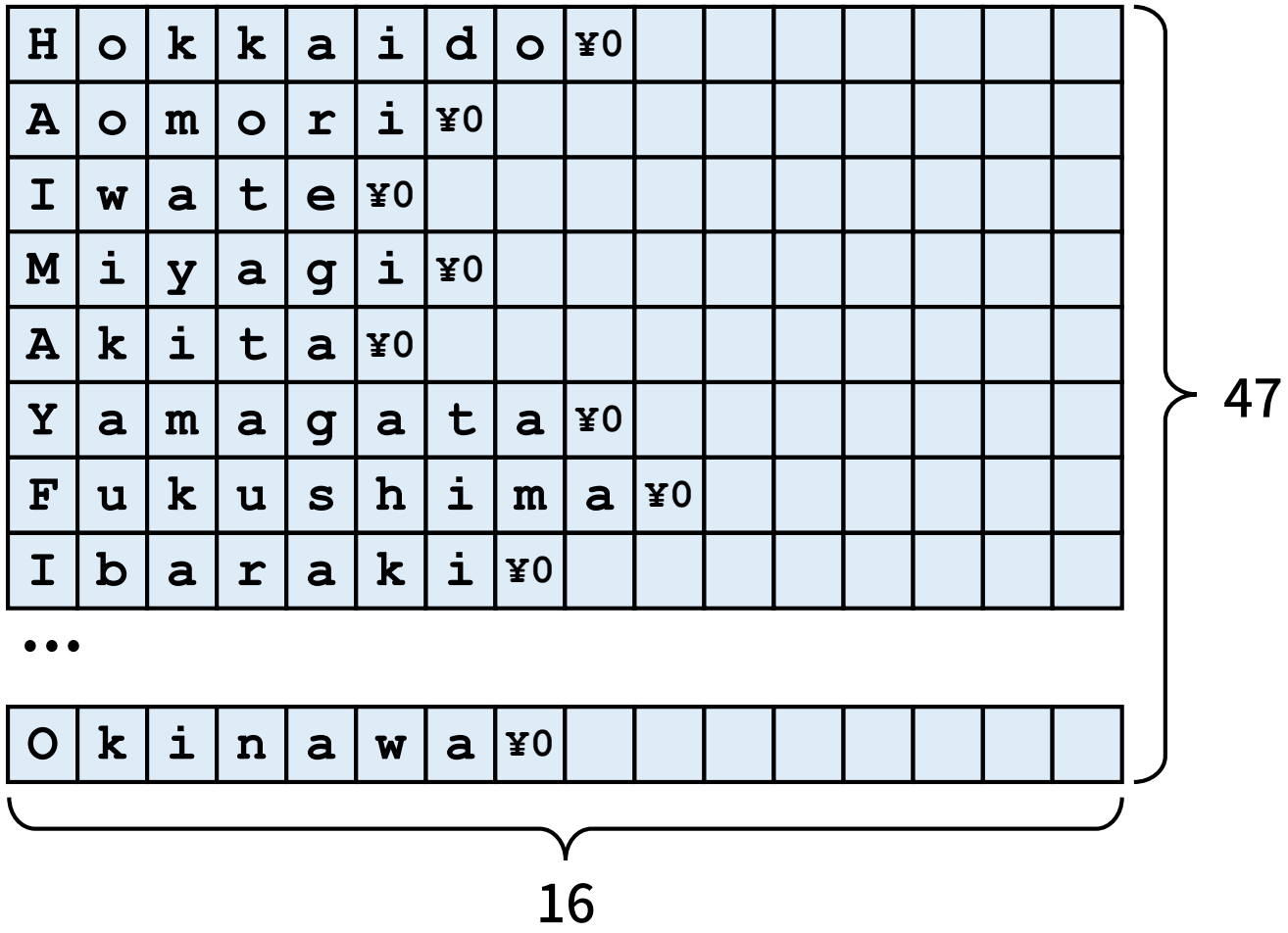
```
char prefecture_names[47][16];
```

- 文字の参照：

  `prefecture_names[i][j]` で，`i` 番目の文字列の `j` 番目の文字を参照できる

文字列の配列の例

prefecture_names



コマンドライン引数（再掲）

コマンドライン引数の使い方

- プログラムにはコマンドライン引数を与えることができる
  - 文字列の並び
  - 例：`$ ./a.out 1 abc 3.4 pqrs`
- プログラムはそれらを `main` 関数の引数として受け取る

```
int main(int argc, char *argv[])
{
   …
}
```

  - `argc` はコマンドライン引数の数
  - `argv` はコマンドライン引数（文字列）の配列
    - 配列の先頭（0番目の）要素はプログラム名

コマンドライン引数を表示するプログラム

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i;
  printf("The number of the arguments is %d¥n", argc);
  for (i = 0; i < argc; i++) {
    printf("The argument %d is %s¥n", i, argv[i]);
  }
  return 0;
}
```

コマンドライン引数からの整数の受け取り

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  int a, b;
  if (argc != 3) {
    fprintf(stderr, "Error: specify two integers.¥n");
    exit(1); /* Terminate this program */
  }
  a = atoi(argv[1]); /* Convert string to integer */
  b = atoi(argv[2]); /* Convert string to integer */
  printf("%d + %d = %d¥n", a, b, a + b);
  return 0;
}
```

オンラインマニュアル

man コマンド

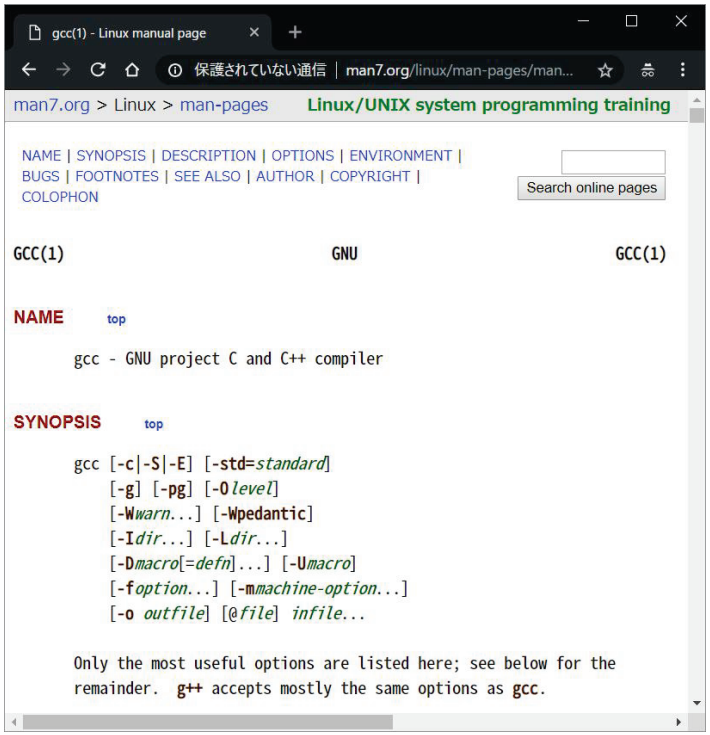
- 端末上でコマンドやライブラリ関数のマニュアルを見ることができる
  - `man strcmp` を実行
    → `strcmp` 関数のマニュアルが表示される
  - コマンドに -h や --help などのオプションを与えると表示されるヘルプよりも大抵詳しい
- セクションの概念があることに注意
  - `man printf` を実行
    → `printf` コマンドのマニュアルが表示される
  - `man 3 printf` を実行
    → `printf` 関数のマニュアルが表示される

オンラインマニュアルのセクション

1. コマンド (プログラム)
2. システムコール
3. ライブラリコール
4. スペシャルファイル (デバイス)
5. ファイルのフォーマットと規約
6. ゲーム
7. 概要、約束事、その他
8. システム管理コマンド

Web 上にあるコマンドや ライブラリ関数のマニュアル

- Linux 用
  - The Linux man-pages project https://www.kernel.org/doc/man-pages/
  - JM Project https://linuxjm.osdn.jp/
  - die.net https://linux.die.net/man/



演習

演習に関するアドバイス

- コンパイラが出す警告やエラーメッセージには貴重な情報がある
  - デバッグのヒントが得られる
  - 警告が出ていても実行可能プログラムは作られるし，それが正しく動くことも多いが，無視しないことを強く勧める
- gcc に -Wall オプションを与えると，より多くの警告を出してくれる
  - -Wextra オプションを与えると，さらに多くの警告
  - 警告オプションをつけることを習慣にしよう
- Mac の端末が固まったら？
  - 別の端末からログインして ~/Library/Caches を削除すると直ることがある
  - ディスク消費量制限（quota）で引っかかっている可能性もある
  - 「教育用計算機システム使用の手引き」をよく読んで下さい

警告の例

```
#include <stdio.h>
int main(void)
{
  int a[2];
  a[0] = 937;
  a[1] = 25;
  a[2] = 601;
  printf("%d, %d, %d¥n",
         a[0], a[1], a[2]);
  return 0;
}
```

```
$ gcc -Wall overrun.c
overrun.c:7:3: warning: array index 2
is past the end of the array (which
contains 2 elements) [-Warray-bounds]
  a[2] = 601;
  ^ ~
overrun.c:4:3: note: array 'a'
declared here
  int a[2];
  ^
overrun.c:9:22: warning: array index
2 is past the end of the array (which
contains 2 elements) [-Warray-bounds]
         a[0], a[1], a[2]);
                     ^ ~
overrun.c:4:3: note: array 'a'
declared here
  int a[2];
  ^
2 warnings generated.
$
```

レポート作成での注意事項（1）

- 作ったプログラムは情報科学類教育用計算機システム（計算機室）の Mac で，gcc コンパイラで動作確認をして下さい
  - 講義で示すプログラムは C89 準拠（のつもり）ですが，レポートで提出するプログラムでは C99, C11, GNU 拡張を使っても良いとします
- 参考： gcc のバージョン
  - 計算機室の Mac（aloeXX）： Apple LLVM version 10.0.1 (clang-1001.0.46.4)
  - 計算機室の Linux サーバ（viola12）： 4.8.5 20150623
  - 全学計算機システムの Linux サーバ（ubuntu.u, icho.u）： 5.5.0 20171010

レポート作成での注意事項（2）

- レポートに書く実行結果には，できる限り，コンパイルする部分も含めて下さい
  - 特に，gcc にコンパイルオプションを与える必要がある場合には，必ず書いて下さい
    - 例えば -std=c11
- プログラムには必ずインデントを付けて下さい
  - 具体的なインデント方式は指定しませんが，プログラム全体で首尾一貫しているようにして下さい
  - そもそも，楽にインデントを付けられるエディタを使って下さい
    - 最近は Atom，Vim，Visual Studio Code，Sublime Text などがよく使われているようです

レポート作成での注意事項（3）

- 指定の書式に沿ったレポートを書いて下さい
- 課題で求められている出力を，課題で求められている処理によらずに出力するプログラムは0点です
  - 課題で求められている出力の文字列を printf の引数に定数として与えるプログラムなど
  - 不正行為とみなして，相応の対処をすることがあります
- メ切厳守
  - manaba の提出用ページが閉じたら終わり
  - メールでレポートを送ってきても，採点しません