iPhone 道場（番外予習：2010年8月18日）

# C プログラミング 即席講座

小嶋　秀樹
xkozima@myu.ac.jp

# C プログラミングの最初歩

Hello, World!

# 例によって Hello World

hello.c

```
#include <stdio.h>

int main (void)
{
    printf("Hello, World!\n");
    return 0;
}
```

「入出力機能を使います」というオマジナイ

メイン関数（これが実行される）

文字列を出力

関数の終了

コンパイルと実行（1）

```
os> ls
hello.c
os> cc hello.c –o hello
os> ls
hello        hello.c
os> ./hello
Hello, World!
os>
```

コンパイルと実行（2）

```
os> ls
hello.c
os> cc hello.c
os> ls
a.out        hello.c
os> ./a.out
Hello, World!
os>
```

# 例によって Hello World

hello.c

```
#include <stdio.h>

int main (void)
{
    printf("Hello, World!\n");
    return 0;
}
```

「入出力機能を使います」というオマジナイ

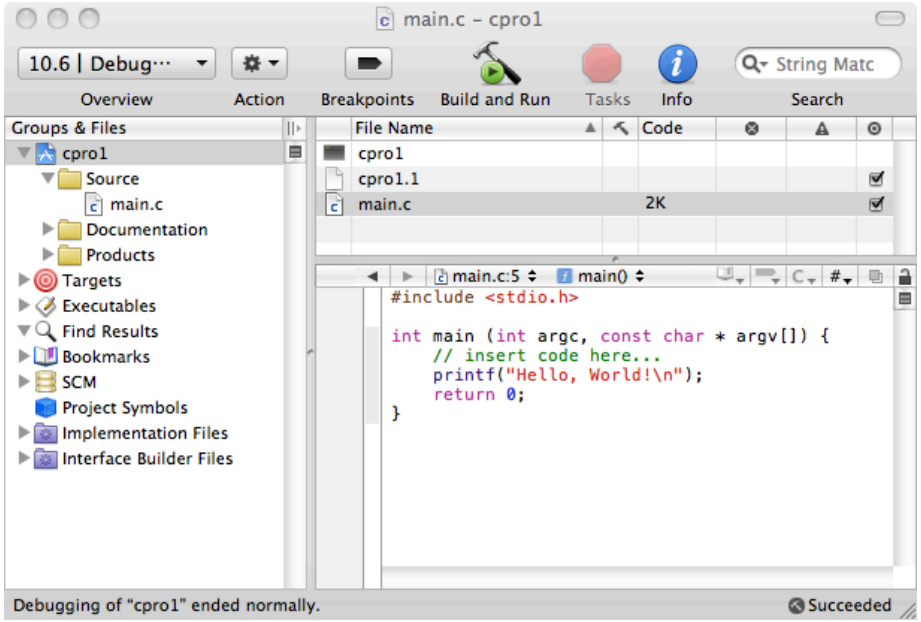
メイン関数（これが実行される）

文字列を出力

関数の終了

Xcode の場合・・・
　Create a new Xcode Project
　Mac OS X / Application
　Command Line Tool
として，プロジェクトを新規作成．

実行結果（入出力）は
　実行 Run ＞ コンソール Console



# 例によって Hello World

hello.c

```
#include <stdio.h>

int main (void)
{
    printf("Hello, World!\n");
    return 0;
}
```

「入出力機能を使います」というオマジナイ

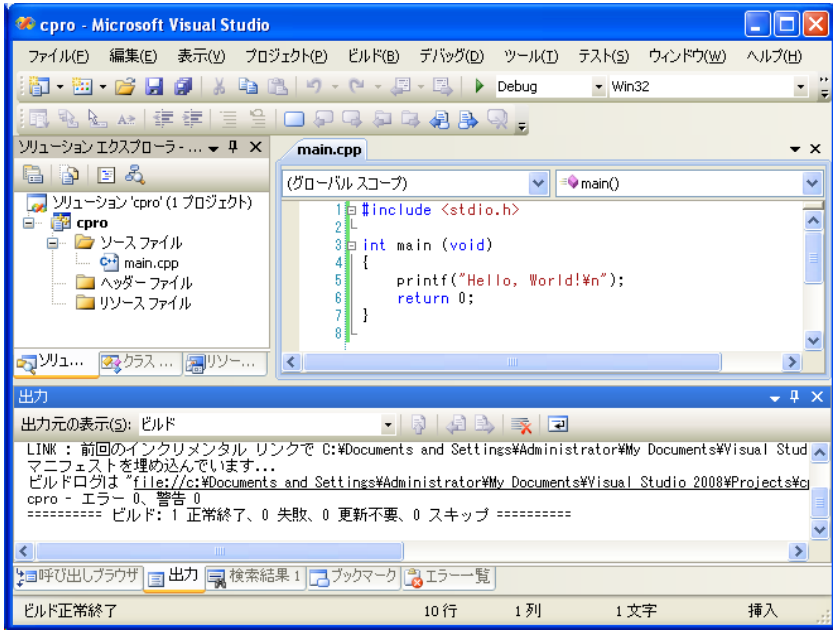
メイン関数（これが実行される）

文字列を出力

関数の終了

Windows の Visual C++ の場合・・・
(Express Edition は無料です)
「空のプロジェクト」を生成
ソースファイル＞追加
＞新しい項目＞C++ファイル

プログラムの実行は，
デバッグ＞デバッグなしで開始



# 例によって Hello World

hello.c

```
#include <stdio.h>

int main (void)
{
    printf(“Hello, World!\n”);
    return 0;
}
```

「入出力機能を使います」というオマジナイ

メイン関数（これが実行される）

文字列を出力

関数の終了

・C のプログラムは関数の集まり（main から実行される）

・関数名・変数名などの「名前」は，
　大文字・小文字を区別する ・・・ Koz と koz は別もの．
　英字で始まり，2文字目以降は英字か数字か “_”．
　予約語（printf など）は関数名・変数名などに使えない．

# 変数と式と代入

## 変数と式と代入

## 整数 int はシンプル

keisan.c

```
#include <stdio.h>

int main (void)
{
    int x;
    int y;
    int wa, sa;

    x = 17;
    y = 6;
    wa = x + y;
    sa = x – y;
    printf(“wa: %d\n”, wa);
    printf(“sa: %d\n”, sa);

    return 0;
}
```

変数宣言
　整数データの入る「箱」が生成

カンマで区切って並べてもよい

“=” は「代入」のこと

文や宣言は “;” で終わる

書式付きプリント
　“%d” に整数データが入る
　“\n” は「改行」の意味

# 変数と式と代入　　実数の扱いは･･･

keisan.c

```
#include <stdio.h>

int main (void)
{
    int ans_i;
    float ans_f;

    ans_i = 17 / 6;
    printf("ans_i: %d\n", ans_i);
    ans_f = 17 / 6;
    printf("ans_f: %f\n", ans_f);
    ans_f = 17.0 / 6.0;
    printf("ans_f: %f\n", ans_f);
    ans_i = 17.0 / 6.0;
    printf("ans_i: %d\n", ans_i);
    ans_f = (float) 17 / 6;
    printf("ans_f: %f\n", ans_f);

    return 0;
}
```

float は実数型
double 倍精度実数もよく使う

整数 / 整数 の結果は整数
"ans_i: 2"

整数 / 整数 の結果は整数
"ans_f: 2.000000"

実数 / 整数 の結果は実数
"ans_f: 2.833333"

実数を整数変数に代入
"ans_i: 2", 切り捨て

(float) 17 とすれば
強制的に 17.0 と解釈する

"%f" には実数データが入る

# 変数と式と代入　　データの種類

```
//  整数(通常は 32bit: -2147483648～2147483647)
int  score;
int  income, outgo;

score = 80;
income = 15000; outgo = 17800;

//  実数
float  height, weight;
double  kousoku;

height = 1.70; weight = 62.0;
kousoku = 299792458.0;          //  299792458 m/s
kousoku = 0.299792458e9;        //  上と同じ値(0.299…×10⁹)

//  文字
char c1, c2;

c1 = 'A';                       //  文字定数 '…'
c2 = 65;                        //  0～255 の ASCII コード
```

'//' から行末まではコメント

'/*' から '*/' まではコメント
(複数行に渡ってもよい)

# 条件分岐とループ

if…else…
for, while

# 条件分岐　if ... / if ... else ...

branch1.c

```
#include <stdio.h>

int main (void)
{
    int n = 12;

    if (n > 10)
        printf("n is greater than 10\n");
    if (n == 0)
        printf("n is zero\n");
    if (n < 10)
        printf("n is less than 10\n");
    if (n % 2 == 0)
        printf("n is even number\n");
    else
        printf("n is odd number\n");

    return 0;
}
```

整数変数を用意し，初期値を与える

"(n > 10) は「条件式」

"==" は「同じ値か」

n % 2 は「2で割った余り」

割り切れれば偶数

そうでなければ奇数

# 条件分岐　if … / if … else …

branch2.c

```
#include <stdio.h>

int main (void)
{
    float height_m  = 1.70;
    float weight_kg = 62.0;
    float bmi = weight_kg / (height_m *height_m);

    if (height_m > 2.5) {
        printf("Use [m] for height.\n");
        return 0;
    }
    printf("Your BMI is %f\n", bmi);
    if (bmi >= 25.0)
        printf("WARNING!\n");
    else
        printf("SAFE\n");
    return 0;
}
```

複数の文を実行するには “{…}” で囲む

25.0 以上の場合

それ以外（25.0 未満）の場合

# 条件分岐　if ... else if ... else if ... else ...

branch3.c

```
#include <stdio.h>

int main (void)
{
    float height_m  = 1.70;
    float weight_kg = 62.0;
    float bmi = weight_kg / (height_m *height_m);

    printf("Your BMI is %f\n", bmi);
    if (bmi >= 30.0)
        printf("You are obese!\n");
    else if (bmi >= 25.0)
        printf("You are overweight.\n");
    else if (bmi >= 18.5)
        printf("You are normal.\n");
    else
        printf("You are underweight.\n");
    return 0;
}
```

もし･･･ならば

それ以外で，もし･･･

それ以外で，もし･･･

どれにも該当しないならば

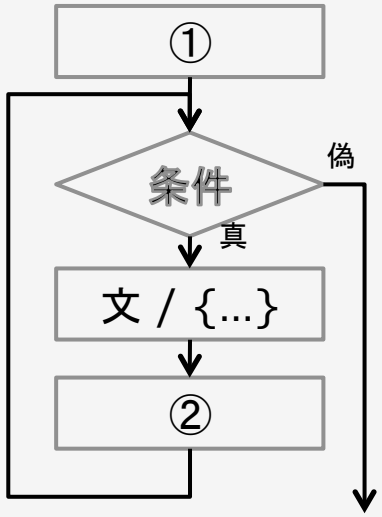
# ループ（繰り返し）　for ループ

loop1.c

```
#include <stdio.h>

int main (void)
{
    int  k, acc = 0;
    int  n = 10;
          ①       条件       ②
    for (k = 1; k <= n; k = k + 1)
        acc = acc + k;
    printf("1+...+%d = %d\n", n, acc);
                                "1+...+10 = 55"
    return 0;
}
```

"k = k + 1" は
"k++" と書いてもよい

"acc = acc + k" は
"acc += k" でもよい




for (①; 条件; ②) 文;

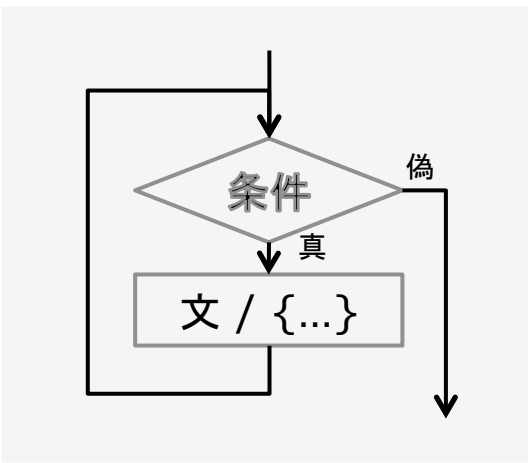
for (①; 条件; ②) {文;...文;}

# ループ（繰り返し）　while ループ

loop2.c

```
#include <stdio.h>

int main (void)
{
    int  k, acc = 0;
    int  n = 10;

    k = 1;
    while (k <= n) {
        acc += k;
        k++;
    }
    printf("1+...+%d = %d\n", n, acc);
    return 0;
}
```

"1+...+**10** = **55**"




while (条件) 文;

while (条件) {文;…文;}

# 条件分岐　　条件式について補足

branch4.c

```
#include <stdio.h>

int main (void)
{
    int  n = 10;

    while (n) {
        printf("%d\n", n);
        if (n > 0) n--;
        else n++;
    }
    printf("BANG!\n");

    return 0;
}
```

「n が 0 以外である限り」

条件式は,
それを評価した結果が
　0以外ならば真
　0ならば偽
となる.

"(n > 0)" を評価すると
　n > 0 のとき 1
　n <= 0 のとき 0
の値をもつ.

# プログラムは関数の集まり

## 関数のつくりかた

# 関数のつくりかた　　関数をつくる・つかう

kansu1.c

```
#include <stdio.h>

int sum (int n)
{
    int k, acc = 0;

    for (k = 1; k <= n; k++)
        acc += k;
    return acc;
}

int main (void)
{
    int ans;

    ans = sum(20);
    printf("1+...+20 = %d\n", ans);
    return 0;
}
```

数学関数 y = f(x) と同じように
- 関数 sum(n) を作る.
- 入力 n は整数型.
- 出力は整数型.

出力を acc として関数を終了

プログラムの実行は main から

引数 20 を与えて関数 sum を呼び出し,
その結果(返値)を ans に代入

main は通常は 0 を返す.
OS がそれを受け取る.

# 関数のつくりかた　関数定義の順序

kansu2.c

```
#include <stdio.h>

int sum (int n);

int main (void)
{
    int ans;

    ans = sum(20);
    printf("1+...+20 = %d\n", ans);
    return 0;
}

int sum (int n)
{
    int k, acc = 0;

    for (k = 1; k <= n; k++)
        acc += k;
    return acc;
}
```

関数本体よりも前にその関数を使用するときは，
こののように，使用する場所でプロトタイプ宣言する．
（つまり{本体}のない関数の（入出力）形式を明示する）

sum を使う関数の中で
プロトタイプ宣言してもよい．
（その関数内部のみで有効）

```
int main (void)
{
    int ans;
    int sum(int n);

    ans = sum(20);
    printf(...);
    return 0;
}
```

int sum(int); と略記可

# 関数のつくりかた　　値を返さない関数

kansu3.c

```
#include <stdio.h>

void bmi_check (float bmi)
{
    if (bmi >= 25.0)
        printf("You are fat.\n");
    else if (bmi >= 18.5)
        printf("You are normal.\n");
    else
        printf("You are thin.\n");
    return;
}

int main (void)
{
    float height_m  = 1.70;
    float weight_kg = 62.0;
    float bmi = weight_kg / (height_m *height_m);

    printf("Your BMI is %f\n", bmi);
    bmi_check(bmi);
    return 0;
}
```

コンソールに文字出力するが
何も値は返さない関数
（出力型に **void** を指定）

何も返さないので return;
(return; を省略してもよい）

main は常に int を返す

# 関数のつくりかた　　値を受け取らない関数

kansu4.c

```
#include <stdio.h>

int RandomValue = 14992;

int random (void)
{
    int tmp = RandomValue * 673 + 944;
    RandomValue = (tmp % 1000000) / 10;
    return RandomValue;
}

int main (void)
{
    int  i;

    for (i = 0; i < 10; i++) {
        int  ran;

        ran = random();
        printf("random() = %d\n", ran);
    }
    return 0;
}
```

地ベタに宣言した変数は
グローバル変数（大域変数）.
どこからでも読み書きできる.

呼び出すごとに
疑似的な乱数 0～99999
を新しく生成する関数

ran は {} の内部で有効な
ローカル変数（局所変数）

```
random() = 9056
random() = 9563
random() = 43684
random() = 40027
random() = 93911
random() = 20304
random() = 66553
random() = 79111
random() = 24264
random() = 33061
```

# 関数のつくりかた　　グローバル変数とローカル変数

kansu4.c

```
#include <stdio.h>

int x = 123;

int square (int x)
{
    x = x * x;
    return x;
}

int main (void)
{
    printf("before: x = %d\n", x);

    int  x = 0;
    while (x < 8) {
        printf("while : x = %d\n", x);
        if (x % 2 == 0) {
            int  x, y;
            x = square(12);  y = square(15);
            printf("if    : x = %d\n", x);
        }
        x++;
    }
    printf("after : x = %d\n", x);
    return 0;
}
```

グローバル変数は，ローカル変数に隠されない限り
どこからでもアクセスできる

関数の引数 "(int x)" は関数内部で定義された
ローカル変数と同じ扱いとなる．
代入もできるが，その結果は外には影響しない．

ローカル変数は，それを宣言した場所から，
所属するブロック {...} の終わりまで有効．

```
before: x = 123
while : x = 0
if    : x = 144
while : x = 1
while : x = 2
if    : x = 144
while : x = 3
while : x = 4
if    : x = 144
while : x = 5
while : x = 6
if    : x = 144
while : x = 7
after : x = 8
```

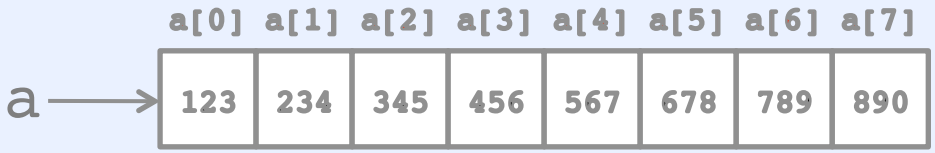
# 配列と文字列

# 配列の宣言と利用

array1.c

```
#include <stdio.h>

int main (void)
{
    int  a[8];

    a[0] = 123;  a[1] = 234;  a[2] = 345;  a[3] = 456;
    a[4] = 567;  a[5] = 678;  a[6] = 789;  a[7] = 890;

    int  i, acc = 0;
    for (i = 0; i < 8; i++) {
        printf("a[%d] = %d\n", i, a[i]);
        acc += a[i];
    }
    printf("mean = %f", (float) acc / 8);

    return 0;
}
```

整数データ8要素からなる配列を宣言




整数変数 acc を実数型に
強制変換して平均値を計算

# 配列の宣言と利用

## 関数に配列を渡すには

array2.c

```
#include <stdio.h>

float array_mean (int x[8])
{
    int i, acc = 0;
    for (i = 0; i < 8; i++)
        acc += x[i];
    return (float) acc / 8;
}

int main (void)
{
    int  a[8];
    a[0] = 123;  a[1] = 234;  a[2] = 345;  a[3] = 456;
    a[4] = 567;  a[5] = 678;  a[6] = 789;  a[7] = 890;

    float  mean
    mean = array_mean(a);
    printf("mean = %f\n", mean);
    return 0;
}
```

整数配列を受け取り，x という名前で参照．
(int x[]) のように要素数不明でもよい

# 配列の宣言と利用

## 関数に配列を渡すには

array3.c

```
#include <stdio.h>

void array_init (int x[8])
{
    x[0] = 123;  x[1] = 234;  x[2] = 345;  x[3] = 456;
    x[4] = 567;  x[5] = 678;  x[6] = 789;  x[7] = 890;
}

int main (void)
{
    int  a[8];
    array_init(a);

    int  i, acc = 0;
    for (i = 0; i < 8; i++) {
        printf("a[%d] = %d\n", i, a[i]);
        acc += a[i];
    }
    printf("mean = %f\n", (float) acc / 8);

    return 0;
}
```

整数配列を受け取り，x という名前で参照．
(int x[]) のように要素数不明でもよい


配列の実体（中身）は同じものを指している

# 文字と文字の配列と文字列

array4.c

```
#include <stdio.h>

int main (void)
{
    char c;

    c = 'A';
    printf("This is the letter %c.\n", c);

    char s[4];

    s[0] = 'M';  s[1] = 'Y';
    s[2] = 'U';  s[3] = '\0';
    printf("This is a string %s.\n", s);
    printf("This is also a string %s.\n", "MYU");

    return 0;
}
```

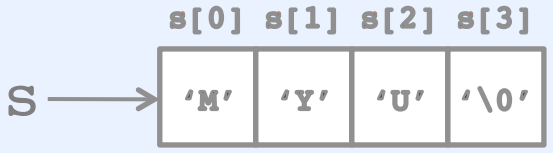
```
This is the letter A.
This is a string MYU.
This is also a string MYU.
```

文字変数（実際には 0～255 の文字コード）

文字定数はシングルクオート '...' で囲む

%c は
1文字と
置き換わる

4要素からなる文字配列

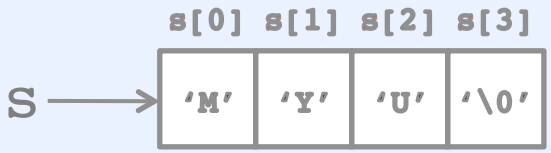



文字配列を「文字列」として扱うとき
'\0' を文字列の終端とする.

# 文字と文字の配列と文字列

array5.c

```
#include <stdio.h>

int main (void)
{
    char s[4] = “MYU”;
    int i;

    for (i = 0; i < 4; i++) {
        if (s[i] == ‘\0’)
            printf(“s[%d] = end\n”, i);
        else
            printf(“s[%d] = ‘%c’\n”, i, s[i]);
    }

    return 0;
}
```




文字配列変数 s に文字列 “MYU” が入る
（配列としての長さは4になることに注意）

if (s[i] == 0) でもよい

```
s[0] = 'M’
s[1] = 'Y’
s[2] = 'U’
s[3] = end
```

# いよいよポインタ

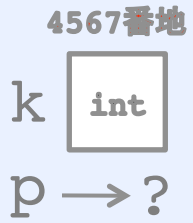
# ポインタ　ポインタの基礎の基礎

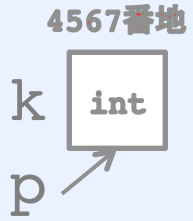
pointer1.c

```
#include <stdio.h>

int main (void)
{
    int k;
    int *p;

    p = &k;

    k = 123;
    printf("k = %d, *p = %d\n", k, *p);
    k = 234;
    printf("k = %d, *p = %d\n", k, *p);
    *p = 345;
    printf("k = %d, *p = %d\n", k, *p);

    return 0;
}
```

p は整数型変数へのポインタ

&k は変数 k のアドレス（メモリの番地）



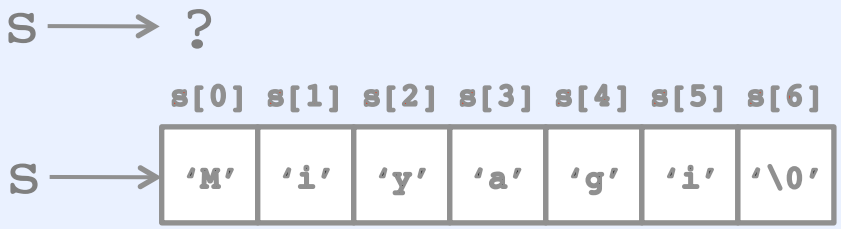

*p はポインタ p が指し示す箱の中身

```
k = 123, *p = 123
k = 234, *p = 234
k = 345, *p = 345
```

# ポインタ　ポインタと配列は似ている

pointer2.c

```
#include <stdio.h>

int main (void)
{
    char *s;

    s = “Miyagi”;

    int i = 0;
    while (s[i] != ‘\0’) {
        printf(“s[%d] = ‘%c’\n”, i, s[i]);
        i++;
    }
    printf(“s[%d] = end\n”, i);

    return 0;
}
```

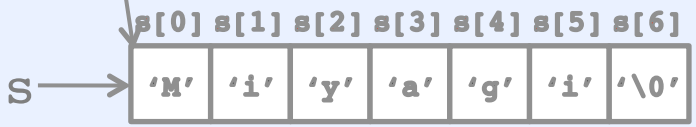



```
s[0] = ‘M’
s[1] = ‘i’
s[2] = ‘y’
s[3] = ‘a’
s[4] = ‘g’
s[5] = ‘i’
s[6] = end
```

# ポインタ　ポインタを関数に渡す

pointer3.c

```
#include <stdio.h>

void explode (char *str)
{
    int i = 0;
    while (str[i] != '\0') {
        printf("str[%d] = '%c'\n", i, str[i]);
        i++;
    }
    printf("str[%d] = end\n", i);
}

int main (void)
{
    char *s;
    s = "Miyagi";

    explode(s);
    return 0;
}
```




プログラムに埋め込まれた文字列

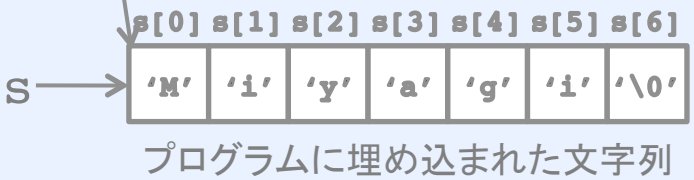
```
str[0] = 'M'
str[1] = 'i'
str[2] = 'y'
str[3] = 'a'
str[4] = 'g'
str[5] = 'i'
str[6] = end
```

# ポインタ　ポインタを関数に渡す（改）

pointer4.c

```
#include <stdio.h>

void explode (char *str)
{
    while (*str != '\0') {
        printf("char = '%c'\n", *str);
        str++;
    }
    printf("str[%d] = end\n", i);
}

int main (void)
{
    char *s;
    s = "Miyagi";

    explode(s);
    return 0;
}
```

```
while (*str) {
    printf("char = %c\n", *str);
    str++;
}
```
と書いてもよい



プログラムに埋め込まれた文字列

```
char = 'M'
char = 'i'
char = 'y'
char = 'a'
char = 'g'
char = 'i'
char = end
```

# 構造体 struct

# 構造体　　構造をもったデータ

struct1.c

```
#include <stdio.h>
struct gp_data {
    float latitude;             // -90..90 北緯は正，南緯は負
    float longitude;            // -180..180 東経は正，西経は負
    float altitude;             // 高度（海抜）[m]
};
void gp_print (struct gp_data *gp)
{
    printf("latitude : %f\n", gp->latitude);
    printf("longitude: %f\n", gp->longitude);
    printf("altitude : %f\n", gp->altitude);
}
int main (void)
{
    struct gp_data  gp_myu;
    gp_myu.latitude  = 38.348337;
    gp_myu.longitude = 140.839502;
    gp_myu.altitude  = 92.50;
    gp_print(&gp_myu);
    return 0;
}
```

*gp は構造体へのポインタ

x->y は，
ポインタ x が指す
構造体のメンバ y

x.y は，
構造体変数 x の
メンバ y

&gp_myu は
構造体変数 gp_myuのアドレス

# 構造体　　構造をもったデータ

struct2.c

```
#include <stdio.h>
#include <math.h>
struct pos {
    double x, y;
};
double distance (struct pos *p1, struct pos *p2)
{
    float dx = p1->x - p2->x,
          dy = p1->y - p2->y;
    return sqrt(dx*dx + dy*dy);
}
int main (void)
{
    struct pos  now, goal;
    now.x  = 1.23;  now.y  = 2.34;
    goal.x = 3.45;  goal.y = 4.56;

    double d = distance(&now, &goal);
    printf("distance = %f\n", d);
    return 0;
}
```

数学関数 sqrt, sin などを使うときのオマジナイ

double sqrt(double); 平方根

現在地点 now, 目標地点 goal

# 構造体　　構造をもったデータ（例）

```
//   視線計測装置 Tobii  から得られるデータ
//   （最高で毎秒120回，このデータが得られる）

struct _TobiiGazeData {
    //   時刻
  int    timestamp_sec;                  //   視線データの取得時刻（秒）
  int    timestamp_microsec;             //   視線データの取得時刻（マイクロ秒）
  //   左目
  float   x_gazepos_lefteye;             //   画面上の視線位置（左目;X）
  float   y_gazepos_lefteye;             //   画面上の視線位置（左目;Y）
  float   x_camerapos_lefteye;           //   Tobiiから見た目の位置（左目;X）
  float   y_camerapos_lefteye;           //   Tobiiから見た目の位置（左目;Y）
  float   diameter_pupil_lefteye;        //   瞳の直径（左目;ミリ）
  float   distance_lefteye;              //   目までの直線距離（左目;ミリ）
  long   validity_lefteye;               //   データ有効性（0=良,1-3=片,4=逸）
  //   右目
  float   x_gazepos_righteye;            //   画面上の視線位置（右目;X）
  float   y_gazepos_righteye;            //   画面上の視線位置（右目;Y）
  float   x_camerapos_righteye;          //   Tobiiから見た目の位置（右目;X）
  float   y_camerapos_righteye;          //   Tobiiから見た目の位置（右目;Y）
  float   diameter_pupil_righteye;       //   瞳の直径（右目;ミリ）
  float   distance_righteye;             //   目までの直線距離（右目;ミリ）
  int    validity_righteye;              //   データ有効性（0=良,1-3=片,4=逸）
};
```

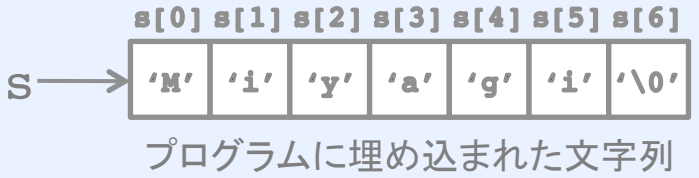
# メモリをイメージする

# メモリのイメージ　　ポインタと配列は似ている？

memory1.c

```
#include <stdio.h>

void ango (char *str)
{
    int i = 0;
    while (str[i] != '\0') {
        str[i] += 1;
        i++;
    }
}

int main (void)
{
    char *s = "Miyagi";

    printf("moto: %s\n", s);
    ango(s);
    printf("ango: %s\n", s);
    return 0;
}
```

← ここで実行時エラーとなる.
（コード領域に埋め込まれた
文字列に書き込もうとしたため）




プログラムに埋め込まれた文字列

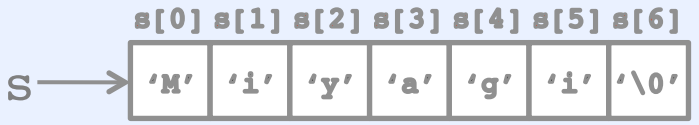
# メモリのイメージ　　ポインタと配列は似ている？

memory2.c

```
#include <stdio.h>

void ango (char *str)
{
    int i = 0;
    while (str[i] != '\0') {
        str[i] += 1;
        i++;
    }
}

int main (void)
{
    char s[] = "Miyagi";

    printf("moto: %s\n", s);
    ango(s);
    printf("ango: %s\n", s);
    return 0;
}
```

← 書き換えてもエラーにならない

7要素からなる文字配列を用意し，
"Miyagi" の各文字で初期化．




```
moto: Miyagi
ango: Njzbhj
```

# メモリのイメージ　4種類のメモリ領域

memory3.c

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

char s_data[5] = “data”;

int main (void)
{
    char *s_code  = “code”;

    char s_stack[6] = “stack”;

    char *s_heap;
    s_heap = malloc(5);
    s_heap[0] = ‘h’; s_heap[1] = ‘e’;
    s_heap[2] = ‘a’; s_heap[3] = ‘p’;
    s_heap[4] = ‘\0’;

    printf(“%s, %s, %s, %s\n”,
           s_code, s_data, s_stack, s_heap );

    return 0;
}
```

コード領域
“code”
（書き込み不可）

データ領域
“data”
グローバル変数

ヒープ領域
“heap”
mallocで切り出し

スタック領域
“stack”
関数呼び出しごとに
ローカル変数を積上げ

```
code, data, stack, heap
```

# メモリのイメージ　　コード領域（テキスト領域）

memory4.c

```
#include <stdio.h>

int main (void)
{
    char *s_code  = “code”;
    printf(“%s\n”, s_code);

    s_code[0] = ‘C’;
    printf(“%s\n”, s_code);

    return 0;
}
```

← ここで実行時エラーとなる.（コード領域に埋め込まれた文字列を書き換えようとした）


コード領域にあるデータは「定数データ」.
書き換えなければ，コード領域にあるデータを使ってよい.

# メモリのイメージ　データ領域（グローバル変数）

memory5.c

```
#include <stdio.h>

char s_data[5] = “data”;

void doSomething (void)
{
    s_data[0] = ‘D’;
}

int main (void)
{
    printf(“%s\n”, s_data);
    doSomething();
    printf(“%s\n”, s_data);
    return 0;
}
```

プログラムのどこからでも
読み書きできる

```
data
Data
```

# メモリのイメージ　スタック領域（ローカル変数）

memory6.c

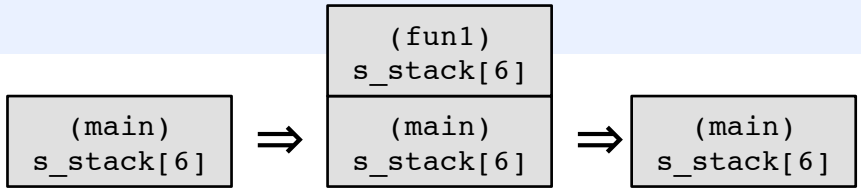
```
#include <stdio.h>

void fun1 (void)
{
    char s_stack[6] = “STACK”;
    printf(“fun1: %s\n”, s_stack);
}

int main (void)
{
    char s_stack[6] = “stack”;
    printf(“main: %s\n”, s_stack);
    fun1();
    printf(“main: %s\n”, s_stack);
    return 0;
}
```

コード領域
“code”
（書き込み不可）

データ領域
“data”
グローバル変数

ヒープ領域
“heap”
mallocで切り出し

スタック領域
“stack”
関数呼び出しごとに
ローカル変数を積上げ




```
main: stack
fun1: STACK
main: stack
```

# メモリのイメージ　スタック領域（ローカル変数）

memory7.c

```
#include <stdio.h>

void fun2 (char *str)
{
    str[0] = 'S';
    printf("fun2: %s\n", str);
}

void fun1 (void)
{
    char s_stack[6] = "stack";
    printf("fun1: %s\n", s_stack);
    fun2(s_stack);
    printf("fun1: %s\n", s_stack);
}

int main (void)
{
    fun1();
    return 0;
}
```

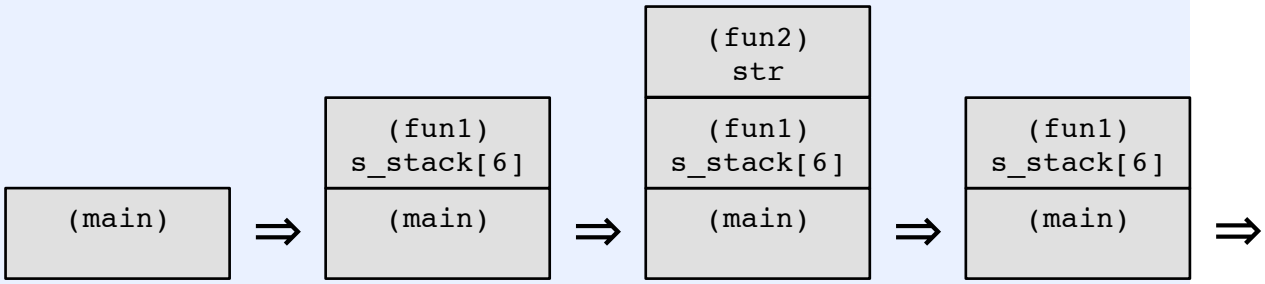
```
fun1: stack
fun2: Stack
fun1: Stack
```

コード領域
"code"
（書き込み不可）

データ領域
"data"
グローバル変数

ヒープ領域
"heap"
mallocで切り出し

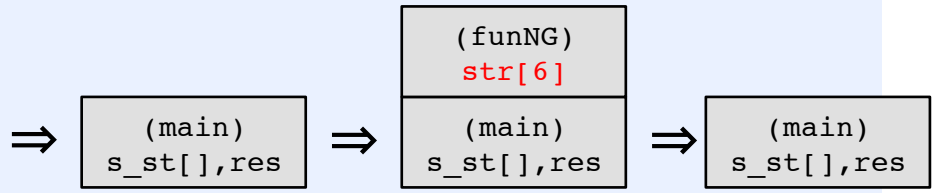
スタック領域
"stack"
関数呼び出しごとに
ローカル変数を積上げ



# メモリのイメージ　スタック領域（ローカル変数） その2

memory8.c

```
#include <stdio.h>

char *funOK (char *str)
{
    str[0] = 'S';
    return str;
}

char *funNG ()
{
    char str[6] = "STACK";
    return str;
}

int main (void)
{
    char s_st[6] = "stack";
    char *res;
    res = funOK(s_st);  printf("%s\n", res);
    res = funNG();      printf("%s\n", res);
}
```




← ローカル変数を呼出し元に返すのは誤り

```
Stack
*error*
```

# メモリのイメージ　ヒープ領域（malloc 切り出し）

memory9.c

```
#include <stdio.h>
#include <stdlib.h>
char *func (void)
{
    char *s_heap;
    s_heap = (char *) malloc(5);
    s_heap[0] = 'h'; s_heap[1] = 'e';
    s_heap[2] = 'a'; s_heap[3] = 'p';
    s_heap[4] = '\0';
    printf("func: %s\n", s_heap);
    return s_heap;
}
int main (void)
{
    char *res;
    res = func();
    printf("main: %s\n", res);
    return 0;
}
```

ヒープ領域に5バイトを確保

ヒープ領域に確保したデータは
関数の呼び出し順序に関係なく
どこからでもアクセスできる.


```
func: heap
main: heap
```

# メモリのイメージ　ヒープ領域（malloc 切り出し）その2

```
//   文字列を格納する領域の確保（ex.文字数が実行時に決まるとき）
char *s;
s = (char *) malloc(3);
s[0] = 'O'; s[0] = 'K'; s[0] = '\0';

//   整数配列を格納する領域の確保（ex.要素数が実行時に決まるとき）
int *a;
a = (int *) malloc(sizeof(int) * 4);
a[0] = 14; a[1] = 15; a[2] = 92; a[3] = 65;

//   構造体を格納する領域の確保
struct pos *p;
p = (struct pos *) malloc(sizeof(struct pos));
p->x = 12.3; p->y = 23.4;

//   不要になった領域を解放
free(s);
free(a);
free(p);
```

確保するバイト数

確保するバイト数

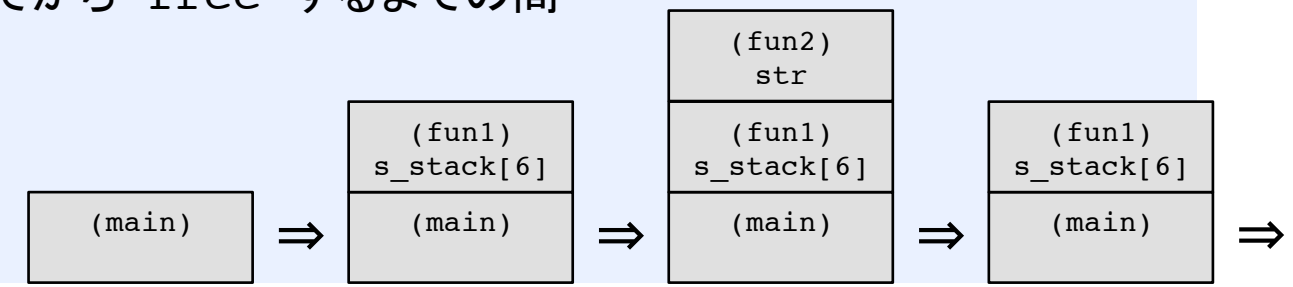
確保するバイト数

malloc で得られたポインタを引数として与える

# メモリのイメージ　　まとめ

```
//   データがどの領域にあるのかをイメージする

char sd[5] = “data”;

void fun (void)
{
     char *sc = “code”;
     char ss[6] = “stack”;
     char *sh = (char *) malloc(5);
}
```


```
//   データにいつアクセスできるか
コード    いつでも可（書き込み不可）
データ    いつでも可（ローカル変数で隠されてなければ）
スタック  スタックの上に載っている間
ヒープ    malloc してから free するまでの間
```



まだまだ序の口ですが...

習うより慣れよ

# 以上です

http://www.myu.ac.jp/~xkozima/lab/mobile-iphone1.html

授業で使った資料（スライドなど）は，
ここからダウンロードできるようにします．