前回の関数のまとめ(1)

復習

- ✓プログラムはmain()関数の先頭から実行される
- ✓関数はmain()関数または他の関数から呼び出されて実行される.
- ✓関数を呼び出す側の実引数の値が関数内の仮引数(変数)にコピーされる
- ✓関数内で求めた値はreturn文によって関数値として呼び出し側に戻される.

関数の定義

スタートポイント

関数の呼び出し

```
#include <stdio.h>

float menseki(float a, float b)
{
        return (a * b / 2);
}

int main(void)
{
        float x, y, kekka;
        printf("底辺は?"); scanf(" %f ", &x);
        printf("高さは?"); scanf(" %f ", &y);

        kekka = menseki(x, y);
        printf("面積は%fです. ¥n", kekka);
}
```

a, b: 仮引数
仮引数もローカル変数
→関数内でのみ有効

x, y:実引数

復習

ローカル変数(1)

```
#include      <stdio.h>

float menseki(float a, float b)
{
        float s;
        s = a * b / 2;
        return s;
}

int main(void)
{
        float a, b, kekka;
        printf("底辺は?"); scanf(" %f ", &a);
        printf("高さは?"); scanf(" %f ", &b);
        kekka = menseki();
        printf("面積は%fです. ¥n", kekka);
}
```

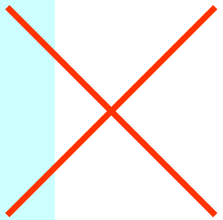
menseki()関数

menseki()関数内のローカル変数aとb

ローカル変数
同じ変数名a,bでも，関数が違うと，別の変数と見なされる
⇒赤色a,bと青色a,bは別の変数

main()関数

main()関数内のローカル変数aとb



引数の値渡し

引数を10倍する関数

復習

main()関数

```
int main(void)
{
    int x = 12;
    mul10(x);
    printf("x = %d", x);
}
```

実引数

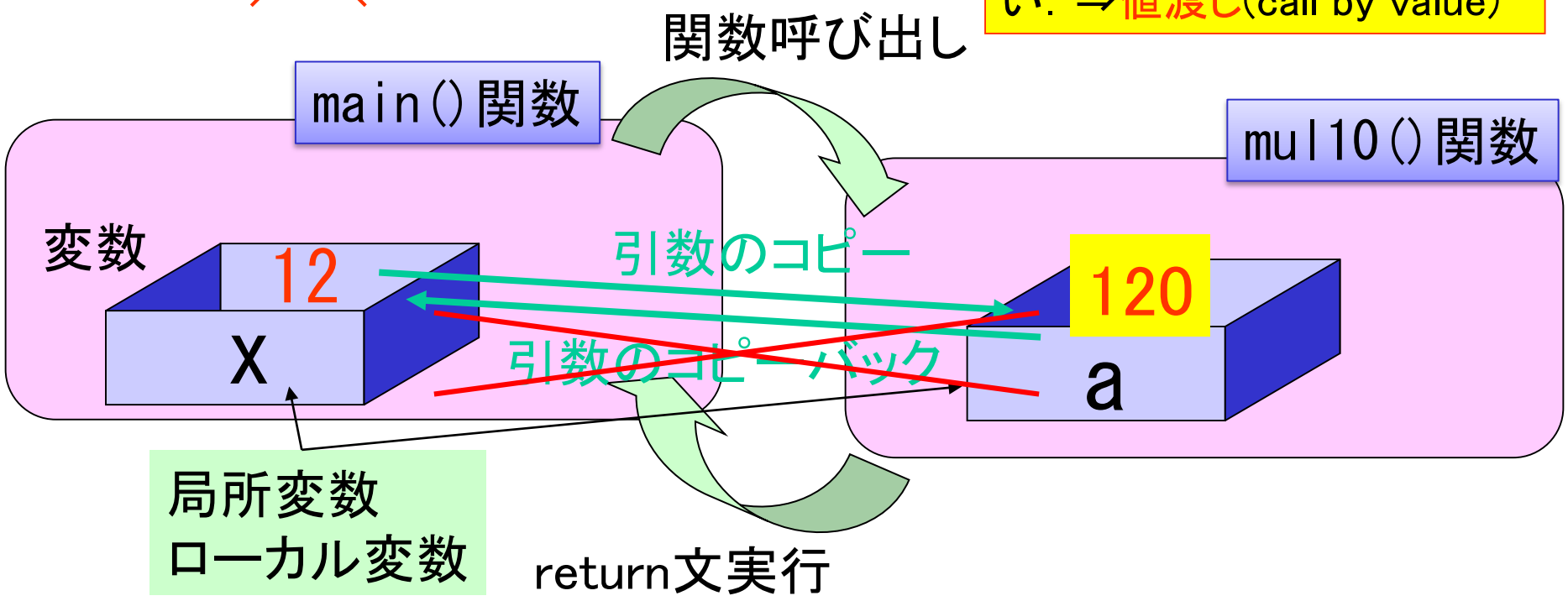
```
x = 12
```

mul10()関数

```
void mul10(int a)
{
    a = a * 10;
    return;
}
```

仮引数

原則 引数は関数からコピーバックされない．呼び出し側の変数(実引数)は変化しない．⇒値渡し(call by value)



配列引数(参照渡し)を用いた関数

復習

n個のデータの値を10倍する関数

```
#inc  e  <stdio.h>

void mul10(int data[], int n)
{
    int i;
    for (i = 0; i < n; i++)
    {  data[i] = data[i]*10;  }
}

int main(void)
{
    int i, aa[30], s;

    // 配列aaに30個のデータを入力する
    for (i = 0; i < 30; i++)
    {
        printf("%d番のデータ:", i);
        scanf("%d", &aa[i]);
    }

    mul10(aa, 30);
    printf("データを全て10倍すると");
    //データの表示(省略)
}
```

配列仮引数:かっこ[ ]のみ付ける. 要素数を空欄にした配列
→data[n]× data[0]× data×

理由:コンパイラから見たら普通の変数に見えるので×

理由:配列全体でなく要素を指定しているので×

データの個数(n個)を引数とする

理由:この関数を再利用することを想定.
データの個数はいつも30個とは限らないので, ループ回数を30回に固定したくない

配列引数は参照渡しであるため, 呼び出し側の配列の値が変化する!

関数内のローカル配列変数dataを変化すると, 呼び出し側の配列aaも変化

配列を実引数とした関数呼び出し
配列名のみ書く.([ ]は不要)

理由1: 配列全体を引数としているのでインデックス付けない
理由2: コンパイラはすでにaaが配列であることを知っている

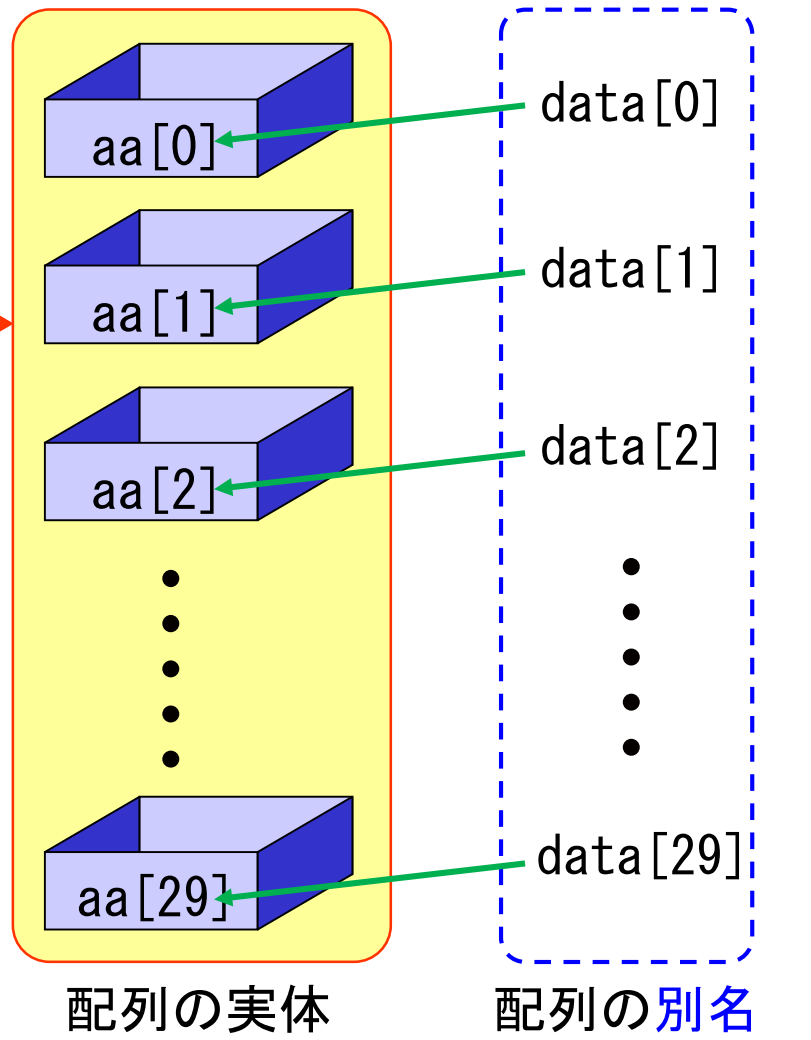
有効なデータの個数は30個

配列引数(参照渡し)における実引数と仮引数

復習

```
#include  <stdio.h>

void mul10(int data[], int n)
{
    int i;
    for (i = 0; i < n; i++)
    {  data[i] = data[i]*10;
}

int main(void)
{
    int i, aa[30], s;

    // 配列aaに30個のデータを入力する
    for (i = 0; i < 30; i++)
    {
        printf("%d番のデータ:", i);
        scanf("%d", &aa[i]);
    }

    mul10(aa,30);
    printf("データを全て10倍すると");
    //データの表示(省略)
}
```

参照渡しでは実引数の値は仮引数にコピーされない



関数のプロトタイプ宣言(1)

```
#include        <stdio.h>

float menseki(float a, float b)
{
    float s;
    s = a * b / 2;
    return s;
}

int main(void)
{
    float kekka;
    kekka = menseki(3, 4);
    printf("s=%f¥n", kekka);
}
```

定義

呼出し

```
#include        <stdio.h>

int main(void)
{
    float kekka;
    kekka = menseki(3, 4);
    printf("s=%f¥n", kekka);
}

float menseki(float a, float b)
{
    float s;
    s = a * b / 2;
    return s;
}
```

呼出し

定義

ソースプログラム上で，関数の定義は関数の呼出しより先に書かれていなければならない

理由:コンパイラはソースプログラムの先頭から順にソースプログラムを機械語に変換するため，関数の定義より先に関数の呼び出しがあると機械語に変換できない.

関数のプロトタイプ宣言(2)

```
#include        <stdio.h>

float func1(...) //何かの関数1
{
    ...
    b = func2(...);
    return s;
}

void func2(...) //何かの関数2
{
    ...
    c = func1(...);
    return;
}

int main(void)
{
    ...
    x = func1(...);
    y = func2(...);
}
```

定義
定義
呼出し
定義
呼出し
呼出し

```
#include        <stdio.h>

void func2(...) //何かの関数2
{
    ...
    c = func1(...);
    return;
}

float func1(...) //何かの関数1
{
    ...
    b = func2(...);
    return s;
}

int main(void)
{
    ...
    x = func1(...);
    y = func2(...);
}
```

関数がお互いに呼び出しあうソースプログラムは記述不可能？？

関数のプロトタイプ宣言(3)

```
#include        <stdio.h>

float menseki(float a, float b);

int main
{
    float kekka;
    kekka = menseki(3, 4);
    printf("s=%f¥n", kekka);
}

float menseki(float a, float b)
{
    float s;
    s = a * b / 2;
    return s;
}
```

関数のプロトタイプ宣言

呼出し

定義

ソースプログラム上で，関数の呼び出し前に，関数の定義か，プロトタイプ宣言のどちらかが絶対必要

printf()関数やscanf()関数では不要？

No！　絶対必要

printf()やscanf()のプロトタイプ宣言はstdio.hファイルに定義されている.

プロトタイプ宣言は関数定義の予告編

プロトタイプ宣言　＝　関数定義の1行目

プロトタイプ宣言は関数の使用方法(引数の数と型，戻り値の型)を示している

ヘッダファイル

関数のプロトタイプ宣言を集めてあるファイル
→ ヘッダファイル（xxxxx.h）
ヘッダファイルをソースプログラム内に読み込む命令
→ #include

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

→ stdio.hという名前のヘッダファイルを読み込む

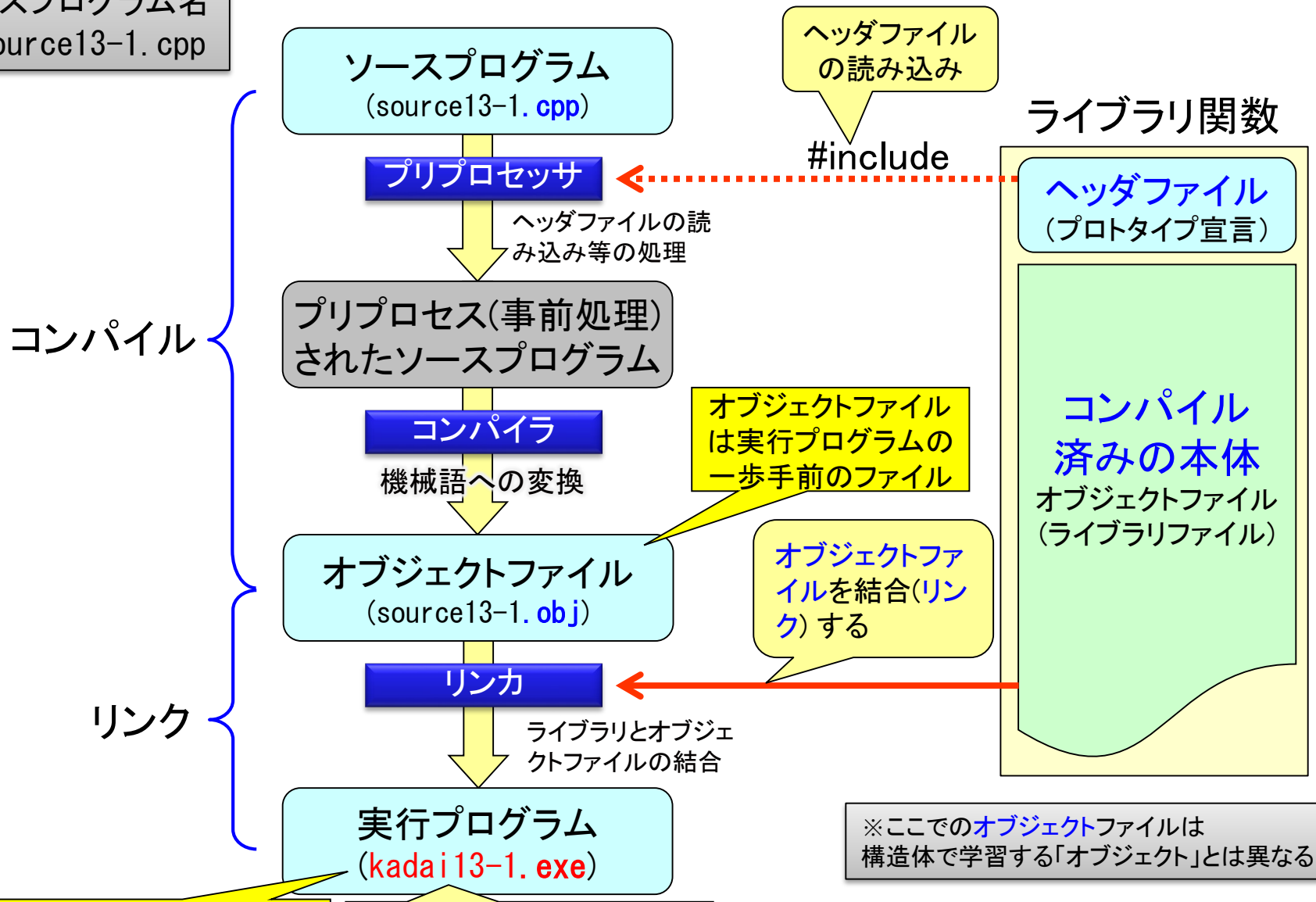
ライブラリ関数

stdio.h
あらかじめ用意されている標準入出力関数であるprintf()やscanf()などのプロトタイプ宣言が書かれているヘッダファイル（standard input output）
math.h
sqrt()などの数値計算関数のヘッダファイル（mathematics）

疑問：ライブラリ関数のプロトタイプ宣言はヘッダファイルに書かれている．では関数本体の定義はどこにある？

実行プログラムができるまで

プロジェクト名
kadai13-1
ソースプログラム名
source13-1.cpp



※ここでのオブジェクトファイルは
構造体で学習する「オブジェクト」とは異なる

.exeファイルだけでプログラムが実行できる！

プロジェクトフォルダ内のDebugまたはReleaseフォルダにできる

コンパイル＋リンク＝ビルド と呼んでいる

プリプロセッサ命令

プリプロセッサ命令は
#で始まる

```
#include   <標準ヘッダファイル名>
#include   "自作ヘッダファイル名"
#define    マクロ名 マクロ文字列
```

→ ヘッダファイルを読み込む

→ マクロ名をマクロ文字列で置き換える

スペース(空白)またはタブ

ソースファイル:kadai12.cpp

```
#include   "rei12.h"
#define    teisu    35
#define    moji     'B'+15
int main(void)
{
    int x,y;
    x = teisu + 22 + moji;
    func1(x);
    y = func2(teisu, moji);
}
```

プリプロセス

プリプロセッサの処理後

```
void func1(int a);
float func2(int a, float b);

int main(void)
{
    int x,y;
    x = 35 + 22 + 'B'+15;
    func1(x);
    y = func2(35, 'B'+15);
}
```

自作ヘッダファイル:rei12.h

```
void func1(int a);
float func2(int a, char b);
```

コンパイル

プリプロセッサ命令を用いたプログラムの例(1)

```
#include    <stdio.h>
#include    <math.h>
#define     PI  3.1415
int main(void)
{
    double x;
    for (x = 0; x <= 2*PI; x = x + PI/10)
    {
        printf("x = %5.3f, sin x = %6.3f¥n",
                                    x, sin(x));
    }
}
```

x=0から2πまでπ/10毎にsin xを計算する

```
x = 0.000, sin x =  0.000
x = 0.314, sin x =  0.309
x = 0.628, sin x =  0.588
x = 0.942, sin x =  0.809
x = 1.257, sin x =  0.951
x = 1.571, sin x =  1.000
x = 1.885, sin x =  0.951
x = 2.199, sin x =  0.809
x = 2.513, sin x =  0.588
x = 2.827, sin x =  0.309
x = 3.142, sin x =  0.000
x = 3.456, sin x = -0.309
x = 3.770, sin x = -0.588
x = 4.084, sin x = -0.809
x = 4.398, sin x = -0.951
x = 4.712, sin x = -1.000
x = 5.026, sin x = -0.951
x = 5.341, sin x = -0.809
x = 5.655, sin x = -0.588
x = 5.969, sin x = -0.309
x = 6.283, sin x = -0.000
続行するには何かキー. . .
```

標準ライブラリ関数を用いたプログラムの例(2)

```
#include    <stdio.h>
#include    <math.h>
#include    <string.h>
#define     PI   3.1415
#define     amp 10
int main(void)
{
    double x, y;
    int p;
    char s[2*amp + 2];
    s[2*amp + 1] = 0;
    for (x = 0; x <= 2*PI; x = x + PI/10)
    {
        _strset(s, ' ');
        y =  sin(x);
        p = (int)(amp * y + amp + 0.5);
        s[p] = '*';
        printf("%s¥n", s);
    }
}
```

21文字の文字列s[]で1文字だけを'*'にして，グラフとして表現する

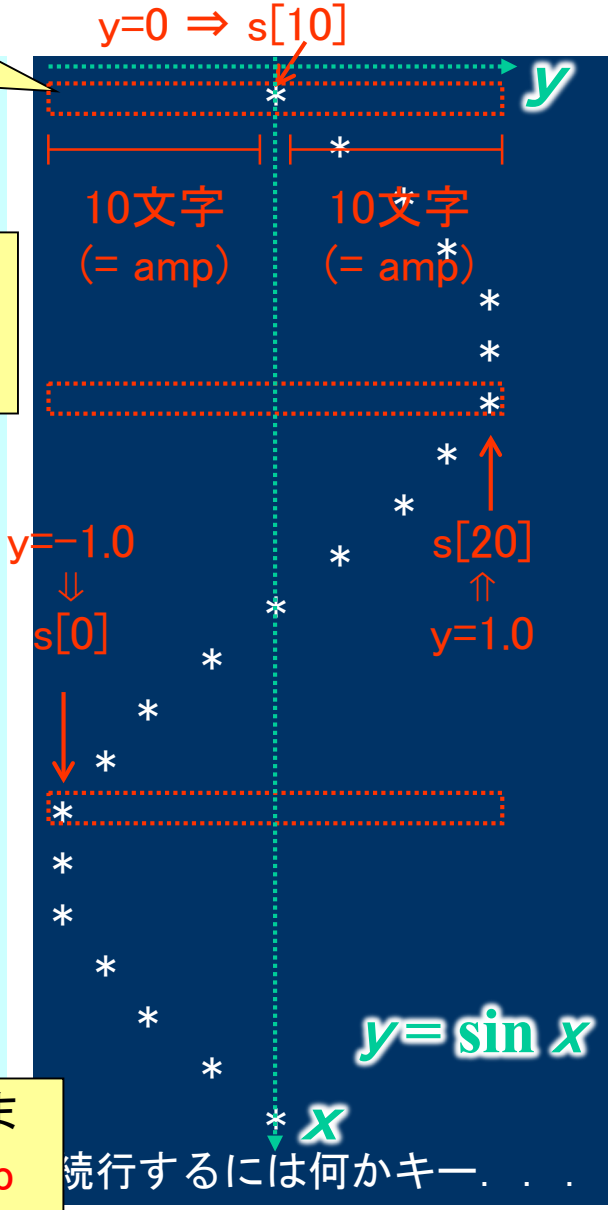
要素数22の配列宣言
注) ampが10に置き換えられてからコンパイルされるので，エラーにならない

両側に10文字ずつ変化する

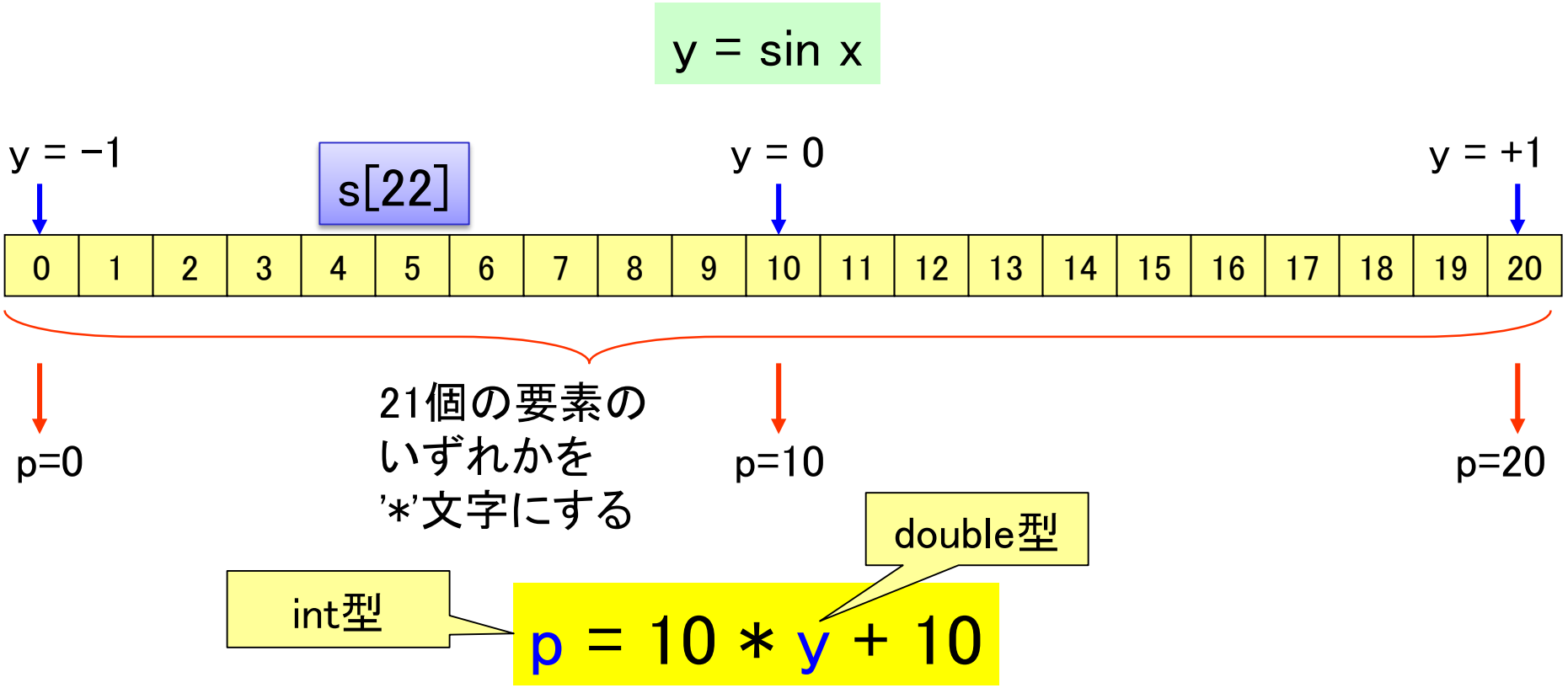
文字列なので最後(S[21])は0にしておく

文字列sを全て特定の文字(ここでは空白文字)で埋めるライブラリ関数

$y = \sin x$のy値から求まる特定のインデックスpの1文字を*に変える



どの文字を'*'に変更するのか？
(変更する文字の位置の求め方の説明)



y：sin(x)の計算結果(double型)

p：文字列sで*印を入れる位置のインデックス(int型)

キャストによる変数型の変換

```
#include    <stdio.h>
int main(void)
{
    int a = 2, b = 3;
    double x1, x2, y1, y2;

    x1 = 1 / 3;
    x2 = a / b;
    printf("x1 =%5.2f, x2 =%5.2f¥n", x1, x2);

    y1 = 1.0 / 3;
    y2 = (double) a / b;
    printf("y1 =%5.2f, y2 =%5.2f¥n", y1, y2);
}
```

```
x1 = 0.00, x2 = 0.00
y1 = 0.33, y2 = 0.67
続行するには何かキー. . .
```

整数同士の演算では少数以下は切り捨て
1/3 = 0.333… ⇒ 0

整数変数同士の演算でも少数以下切り捨て

浮動小数点と整数の演算では全て浮動小数点として計算
1.0/3 = 1.0/3.0 ⇒ 0.333…

整数変数同士の演算だが，浮動小数点で計算させたい

キャストによる型の変換

キャストの書き方
(変数型) 変数
変数を変数型に変換する

キャストを用いた四捨五入

```
#include    <stdio.h>
int main(void)
{
    int c0, c1, c2, c3, c4;
    double x1, x2;

    x1 = 3.4999;
    c0 = x1;
    c1 = (int) x1;

    x2 = 3.5000;
    c2 = (int) x2;
    printf("c1 = %d, c2 = %d¥n", c1, c2);

    c3 = (int) (x1 + 0.5);
    c4 = (int) (x2 + 0.5);
    printf("c3 = %d, c4 = %d¥n", c3, c4);
}
```

```
c1 = 3, c2 = 3
c3 = 3, c4 = 4
続行するには何かキー. . .
```

warning C4244: '=' : 'double' から 'int' への変換です。データが失われる可能性があります。

キャストを用いているので警告が出ない！

キャストを用いた四捨五入
0.5を足して整数型に変換すると小数以下の四捨五入になる

3.4999 + 0.5 = 3.9999 ⇒ 3

3.5000 + 0.5 = 4.0000 ⇒ 4

標準ライブラリ関数を用いたプログラムの例(2)

21文字の文字列s[]で1文字だけを'*'にして，グラフとして表現する

```
#include    <stdio.h>
#include    <math.h>
#include    <string.h>
#define     PI   3.1415
#define     amp 10
int main(void)
{
    double x, y;
    int p;
    char s[2*amp + 2];
    s[2*amp + 1] = 0;
    for (x = 0; x <= 2*PI; x = x + PI/10)
    {
        _strset(s, ' ');
        y =  sin(x);
        p = (int)(amp * y + amp + 0.5);
        s[p] = '*';
        printf("%s¥n",
    }
}
```

p = 10 * y + 10

↓

p = (int)(10 * y + 10 + 0.5)