

プログラミング発展

2023年度2Q 火曜日5~6時限(13:45~16:30)
金曜日5~8時限(13:45~16:30)

工学院 情報通信系

尾形わかは, 松本隆太郎,
Chu Van Thiem, Saetia Supat
TA:東海林郷志, 千脇彰悟

(最終更新 : 6月19日11 : 00)

「構造体(2)」

1. sizeofと符号無し型の落とし穴
2. 構造体を使った総合的課題
～構造体＋これまでに学んだことを組み合わせて～

sizeof演算子

- sizeof は型が占有するバイト数または変数（配列や構造体含む）を占有するバイト数をsize_t型で返す演算子（／や！のようなもの）である。
- 「sizeof(型名)」および「sizeof 変数名」として使う。
- size_t型は符号が無い整数型としてstddef.h で定義されている。実際にはunsigned longまたはunsigned long long

配列の要素数 (sizeof の用途)

- `float array[128]; ... for(i=0; i<128; i++) { 配列の各要素への処理 }` のように、同じになるべき値（ここでは配列の要素数）を2回以上書くと、変更したときに一部を更新しわすれてバグになりやすい
- 配列の要素数は `sizeof array / sizeof array[0]` と表現すれば更新し忘れによるバグを防ぐことができる
- 配列の要素数自体は `#define` で定義するべきだが、定義済みの配列の要素数の名前を思い出すのは（配列が数百種類あると）面倒だし、配列要素数名前を変更したときにも変更せずに済むため、配列要素数を `sizeof` で参照すると便利である。
- 大規模プログラムではプログラム内で定義された名前が膨大にあるため、配列名と配列要素数名を両方覚えるよりは、配列名だけ覚えて作業するほうが楽。

符号無し整数の落とし穴

- 符号付き整数（int型等）と、符号無し整数（unsigned int型等）を両方含む式があると、結果がおかしくなることがある。
（具体例は次ページ）
- これは、C言語における型変換のルールによる

符号無し整数の落とし穴の例

unsigned2 > unsigned2 > C main > main()

```
1 #include <stdio.h>
2
3 int main(void) {
4     int i=-1; unsigned u=0; long long ll;
5     ll=i+u;
6     printf("ll=%lld\n", ll);
7     return 0;
8 }
9
```

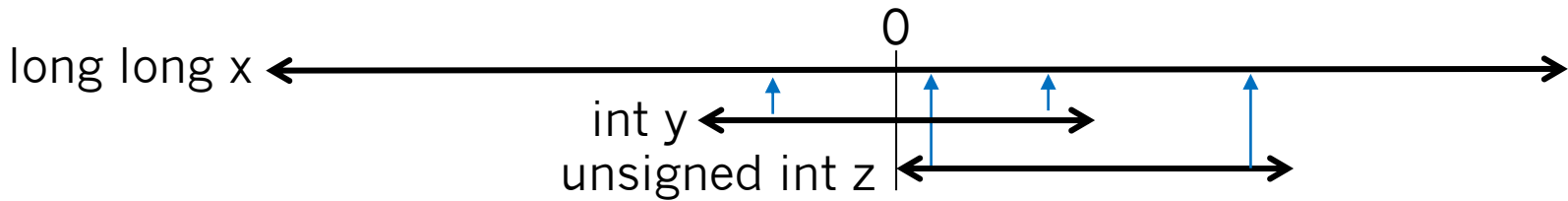
```
ll=4294967295
Program ended with exit code: 0
```

型変換のルール（抜粋）

- 異なる整数型を混ぜ合わせた二項演算（四則演算や比較）では変数型の暗黙の変換が起きる
- 二項演算の片方の型が他方の型の表現範囲を含む場合には、表現範囲が広い型に変換された後に式が評価され、不都合が起きない（この表現は厳密ではない...）
- そうではない二項演算では符号無し型の一番大きい型に変換される
- 二項演算に浮動小数点が含まれると浮動小数点に変換されてから評価される
- 符号無し→有り や 符号有り→無し の型変換でも、変換前の数値が変換後の型の表現範囲に収まる場合には値が変わらない

型変換の例 1

- Case 1:

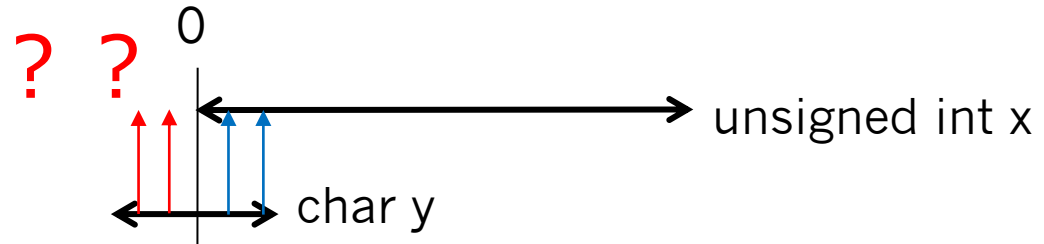


long long はintとunsigned int の表現範囲を含むので
 $x+y+z$ [$(x+y)+z$ と等価] の計算においては、
すべてlong long に変換され、不都合は起こらない

※ $z+y+x$ [$(z+y)+x$ と等価] と書くと、まず y が unsigned intに変換されるため、次ページの例と同様に不都合が起こる可能性がある

型変換の例 2

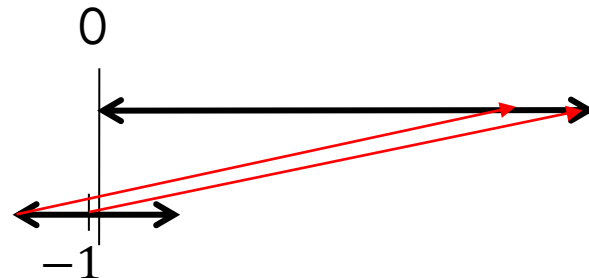
- Case 2:



char と unsigned int は、互いに表現範囲を含まないので
if ($y > x$) の比較においては
yが **unsigned int** に変換される

y < 0 の場合にどう変換するかは**未定義**
($y \geq 0$ のときにはunsigned変換後も値は変わらない)

良く見られる例：
-1が符号なしの最大値に



C言語の型変換のルールは複雑怪奇

なので詳しく知りたい人は、まともな参考書またはC言語の規格そのものを読んで下さい（参考書を選ぶときに、暗黙の型変換をきちんと説明していないものはよくないので避けるべき。また自動変数と静的変数もきちんと説明があるべき）

指定参考書の訂正

5 / 2 = 2 ← 小数点以下は切り捨てられている

2 型変換

算術演算を行うとき、2つのオペランドの型が異なる場合は、自動的に暗黙の型変換 (implicit conversion) が行われます。以下の通常の算術型変換 (usual arithmetic conversion) の規則に従って型が変換され、演算の結果もこの型になります。

- ①一方のオペランドが long double 型であるとき、他方のオペランドは long double 型に変換されます。
- ②上記でなく、一方のオペランドが double 型であるとき、他方のオペランドは double 型に変換されます。
- ③上記でなく、一方のオペランドが float 型であるとき、他方のオペランドは float 型に変換されます。
- ④上記でないときは、両方のオペランドは整数であり、以下の規則が適用されます。
 - (I) 両方のオペランドが同じ型であるときには、変換の必要はありません。
 - (II) 上記でなく、一方のオペランドが符号付き整数型であるか、一方のオペランドが符号なし整数型であるときは、より精度の高い型にあわせて変換されます。
 - (III) 上記でなく、一方のオペランドが符号なし整数型であって精度が他方のオペランド以上である場合、符号付き整数型のオペランドは符号なし整数型に変換されます。

指定参考書の次ページ

第3章 演算子

(IV) 上記でなく、一方のオペランドが符号付整数型であり、他方の符号なし整数型のオペランドの値をすべて表現できるとき、符号なし整数型のオペランドは符号付整数型に変換されます。

つまり、精度が高く、範囲の広いほうに変換されるのです。なお、算術型から `_Bool` 型に変換する場合は、算術型の 0 は `_Bool` 型では 0、算術型の 0 以外の値は `_Bool` 型では 1 に変換されます。

例2 異なった型の算術演算

`int` 型、`double` 型をオペランドとして、除算を行ってみましょう。

プログラム

```
/*  
  異なった型の算術演算  
*/
```

整数型をどう扱うか 1

- **符号無しと符号有りを混ぜない**

もし止むを得ず混ぜるしかない場合には

- すべての型の表現範囲を含む整数型に全変数を型変換してから演算する

しかし、size_tの表現範囲と符号有り整数の表現範囲両方を含む整数型は存在しないことが多い（困った...）。そういう場合には次ページのように対応できる

整数型をどう扱うか 2

符号有り整数型typeSの変数sを符号無し整数型typeUの変数uを扱うときに

1. uがtypeSの表現範囲に収まるならuをtypeSに変換してから演算する (例 和 = $s + (\text{typeS})\ u$ 、ここで(typeS)は型変換演算子)
2. sがtypeUの表現範囲に収まるならsをtypeUに変換して処理する (例 和 = $(\text{typeU})s + u$)
3. 1も2も満たされない場合の対応はケースバイケース

各整数型の表現範囲

#include <limits.h>すると

https://ja.wikipedia.org/wiki/%E6%A8%99%E6%BA%96C%E3%83%A9%E3%82%A4%E3%83%96%E3%83%A9%E3%83%AA%E6%95%B4%E6%95%B0%E5%9E%8B%E3%81%AE%E5%A4%A7%E3%81%8D%E3%81%95_limits.h

で説明される最大値・最小値をの名前
(INT_MAXなど) が定義される

printfのsize_t型指定文字

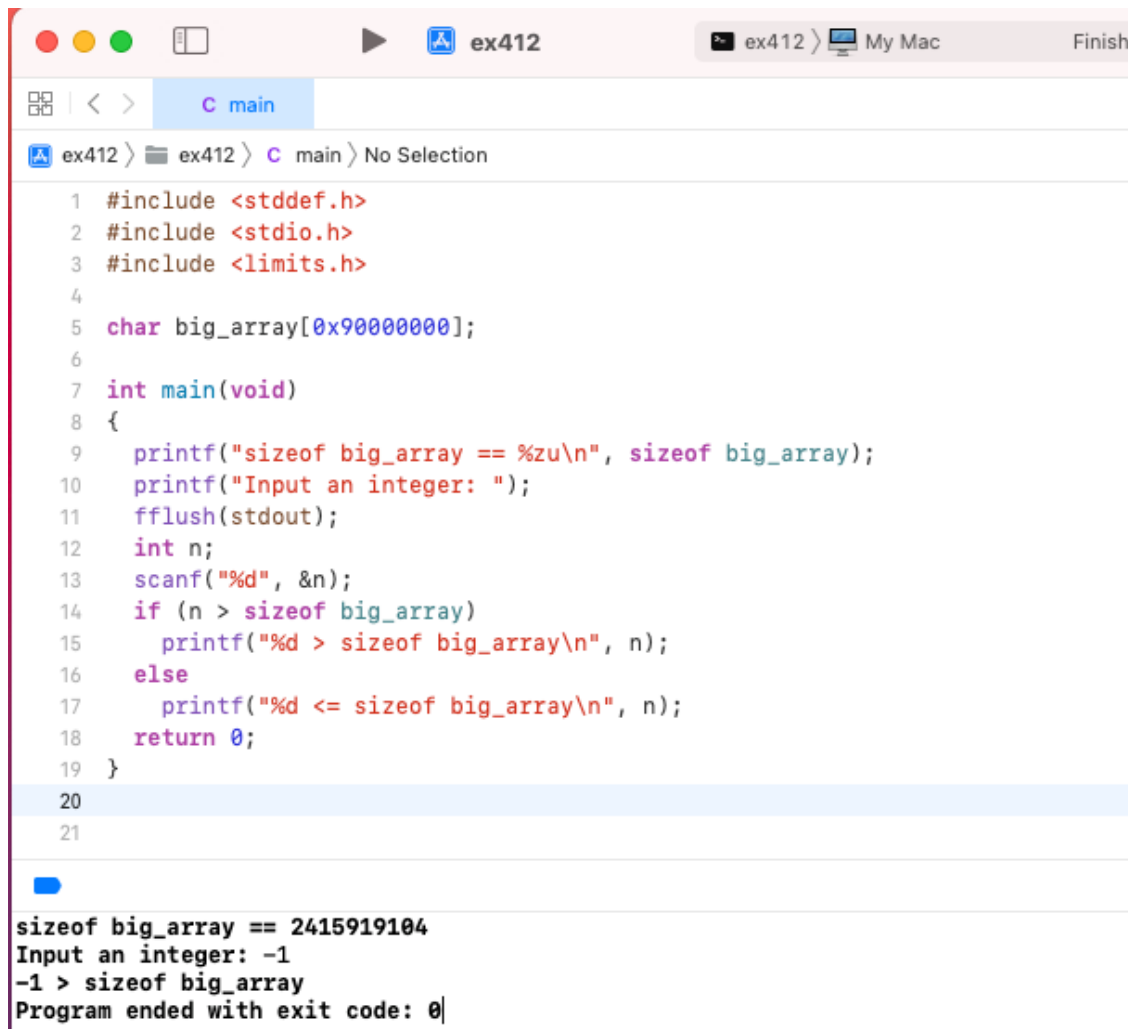
- printf で数値を表示するときに数値の型を正しく指定しないと表示がおかしくなることがあるため、intとfloatとdouble型以外の引数には必ず型指定文字を付ける（1回目資料の再掲）
- **size_tであるときにはzを付ける**（例えば「%zu」や「%zx」であり、「%zd」は間違いなので注意）。
- **型指定文字が正しく付いていない提出物は減点する可能性がある**（1回目資料の再掲）
- %の後の型指定文字についてはウィキペディアに説明がある

<https://ja.wikipedia.org/wiki/Printf#%E9%95%B7%E3%81%95%E4%BF%AE%E9%A3%BE%E5%AD%90>

Ex4-1 (size_t型との闘い)

以下のプログラムでは配列big_arrayの大きさと入力された整数値の比較に失敗している。入力値がint型の表現範囲に収まるときにどのように修正すればよいか述べて。

提出物: ex4_1.pdf



```
1  #include <stddef.h>
2  #include <stdio.h>
3  #include <limits.h>
4
5  char big_array[0x90000000];
6
7  int main(void)
8  {
9      printf("sizeof big_array == %zu\n", sizeof big_array);
10     printf("Input an integer: ");
11     fflush(stdout);
12     int n;
13     scanf("%d", &n);
14     if (n > sizeof big_array)
15         printf("%d > sizeof big_array\n", n);
16     else
17         printf("%d <= sizeof big_array\n", n);
18     return 0;
19 }
20
21
sizeof big_array == 2415919104
Input an integer: -1
-1 > sizeof big_array
Program ended with exit code: 0
```

Ex4-1へのコメント

- sizeof big_arrayの大きさはint型最大値を超えているため、(int)sizeof big_arrayは負の値となる
- 前ページのプログラムはT2Scholaにある
- 変更方針を実際にプログラムに反映して動作確認することが望ましいが、big_arrayが巨大すぎるため松本研究室のM1 Mac（と尾形先生のWindows Visual Studio 2022）では動作しなかった。動作しない場合は配列の要素数を（思い切って）減らすとよい。計算機室のIntel Macでは変更なしに動作した。
- big_arrayの要素数を変更（増加および減少）した場合および変更しない場合すべてで正しく動作する方針を示すこと。従ってbig_arrayの大きさを数値としてプログラムに記入できず、sizeof big_arrayで参照する必要がある。
- big_arrayの大きさは、個人向けに販売されているコンピュータの主記憶（メモリ）に収まる程度だと仮定してよい。

Ex4-2 (構造体のまとめ)

～構造体＋これまでに学んだことを組み合わせて～

T2Scholaに東京都の郵便番号一覧のCSVファイル"test_dat.txt"(*)がある．これをワークスペースのプロジェクトのフォルダにコピーする．

CSVファイルのフィールドは、郵便番号、県名、市区名、町名等

(番地よりも上位) から成る (テキストファイルとして読める；各フィールド毎の文字列は"で囲われている)．これを読み込んで、

- ・郵便番号から住所を検索

することを考える．

1) csvファイルを読み込んで、構造体の配列に格納するプログラムを以降の頁を参照して、作成せよ．

練習用に、千代田区分だけ抜き出したファイル"chiyoda.txt"

(60件強) を用意してある (ので、最初はこれを読み込んでみよ) ．

提出プログラムでは test_dat.txt を読むこと

* JPからダウンロードできるCSVファイルから抜粋(日本語を削除；東京都の部分だけ抜粋
高層ビルのフロアごとにつけられた番号を除いてある；1700件強 2016年度のもの)

データの構造体は、例えば、下のようものを想定

```
typedef struct _ziptable {  
    int zip;  
    char pref[PREF_STR_SIZE]; // ← 15文字程度あれば十分  
    char city[CITY_STR_SIZE]; // ← 30文字程度あれば十分  
    char addr[ADDR_STR_SIZE]; // ← 50文字程度あれば十分  
} ziptable_t;
```

これは、構造体のデータ型を定義しただけなので、実際の変数（配列）としては

```
ziptable_t ziptable[2000]; // ← 読み込むデータ量に応じて変更が必要
```

などと宣言しないといけない

CSVファイルからデータを読み込むには下記を参照せよ.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
:
int main(void) {
    ziptable_t ziptable[ZIPTABLE_MAX];
    FILE *infp;
    clock_t start, end;
    int k, n;
    int key, pos;
    if ( ( infp = fopen("test_dat.txt", "r")) == NULL) {
        fprintf(stderr, "File not found¥n");
        exit(EXIT_FAILURE);
    }
    /*      read csv file          */
    n = read_from_csv(ziptable, infp);
    printf("number of data = %d¥n", n);
    fclose(infp);
}
```

main()から用いられる関数群を T2Scholaに“zip202?.c” として掲載しています。
郵便番号データと合わせて単一のzipファイルになっています。

Ex4-2 (続き)

2) バブルソート**など**により構造体の配列を郵便番号順にソートせよ。
ソート結果から、等間隔にいくつかのデータを抽出して、
郵便番号とそれに対応する県名、市区名、町名等を入力し、
結果が正しくソートされていることを確認せよ。標準関数qsortの
使用を認める

★提出物：(データを読み込み、ソートし、確認結果を出力する)
3) のプログラムに含むようにして出力する。

3) 2) のソートの実行時間を計測せよ。

★提出物：2) のプログラム(ソート後の確認結果を出力する)
"ex4_2_2.c"

Ex4-2 (続き)

4) 2) のプログラムを拡張し**ソート済み**のデータについて郵便番号を検索して、県名、市区名、町名等をそれぞれ出力する**.

キーとなる郵便番号nnnnnnnnは標準入力からscanf()を用いて与え、探索方法は何でもよい。出力は、

nnnnnnnn : 県名, 市区名, 町名等

の形式として、キーの郵便番号が見つからなかったときは、

nnnnnnnn : Not Found

を出力せよ。

C言語標準関数のbsearch (二分探索) の使用を認める(使わなくても可)

★提出物： 4) のプログラム

(標準入力から入力した郵便番号をキーとして、それぞれの検索結果(郵便番号、県名、市区名、町名等)を出力する)

“ex4_2_4.c”

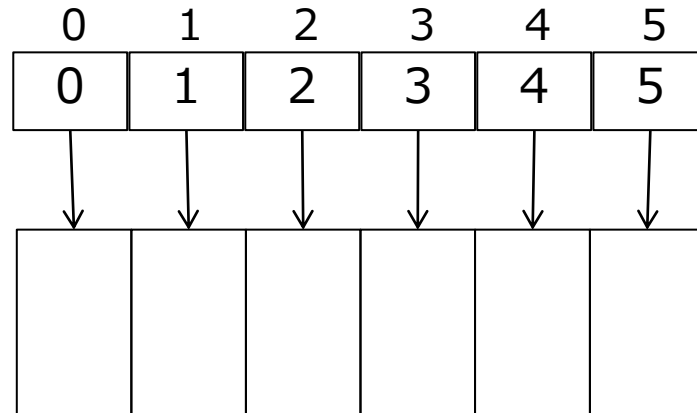
** 同じ郵便番号で異なる住所や同じ住所で郵便番号が異なる場合があるが、1つ出力でよい

EX4-2

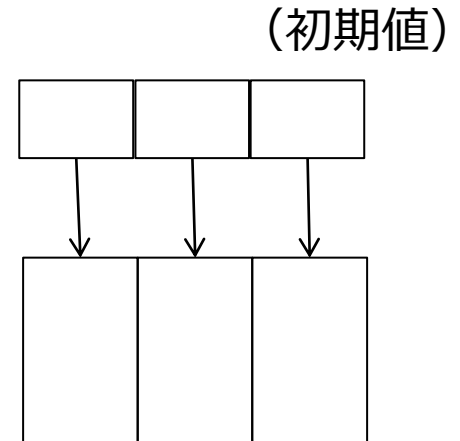
(続き、小問5以降は発展的課題であり提出すれば加点する)

- 5) 各データの構造体配列における位置 (インデックス) を与える
下記のような配列を考える.

配列の
インデックス
配列の値



...



...

構造体
(配列)

このインデックス配列をソート時に交換 (構造体ごと交換しない) することにより, データの並び換えを行うように2) のソート方法を書き換えよ. ソート結果から, ほぼ等間隔に10個程度のデータを抽出して, 郵便番号とそれに対応する県名, 市区名, 町名等を出力し, 結果が正しくソートされていることを確認せよ.

Ex4-2 (続き)

6) 5) のソートの実行時間を計測せよ。

★提出物： 5) のプログラム(ソート後の確認結果を出力する)

"ex4_2_5.c"

7) 3) と 6) で求めた実行時間を比較して、実行時間の違いの原因を考察せよ。 3) と 6) の実行時間は 7) の提出物だけに記入すればよい。

★提出物： 7) の考察

"ex4_2_7.pdf"

注意：小問2～6ではすべて**同一のソート方法**（バブルソート、クイックソート等）を用いること。**C言語標準関数の qsort の使用を認める（使わなくてもよい）**。ただし標準関数のqsortを用いるなら2～6すべてで用いること。時間の差が3) と 6) で少なすぎる場合、1 2 万件以上ある全国郵便データを使って比較してもよい。

実行時間が短すぎる場合

- 郵便件数を多くすれば実行時間が伸びるが、T2Scholaに全国郵便番号を含むファイル KEN_ALL_ROME.txt があるのでそれを使うと良い
- 上記データは余りにも巨大で自動変数の配列に読み込むとメモリ不足になるかも知れない。その場合静的変数の配列（3回目講義）またはmalloc()（6回目講義）を用いるとよい
- 自動変数へのメモリ制限を解除していくらでも使えるようにすることはできるが、環境依存で採点に困るのでその方法はやめて下さい

fopen()の使い方

- fopen()の第 1 引数はいわゆるパス名であることを後の回で説明する
- パス名の説明はT2Scholaの「参考資料」にある
- 採点の都合上、提出プログラムのfopen()第一引数には**フォルダ名・ディレクトリ名を含めない**でください
- fopen()でファイルを開けない場合相談して下さい

Ex4-3 (文字列と16進数の復習 2)

1. **unsigned** int n;と宣言した**符号無し**整数型変数に scanf("%x",&n);で非負整数を読み込み、
2. char string[128]; と宣言した配列に16進数の文字列に変換して格納し、
3. printf("¥n%s¥n", string); によって出力し終了するプログラムを作成せよ

注意事項など：

- printf()とscanf()とfflush()以外に開発環境が提供する関数を用いないこと
- 16進数のa～fは小文字でも大文字でもどちらでもよい
- 提出ファイル： ex4_3.c

%xについて（再掲）

- printfやscanfで整数を16進数で表示・入力する%xは、対応する整数型が符号無しであることを要求し、符号有り整数を用いた場合の動作は規格上では未定義である
- %xに対応する変数に符号有り変数を与えても、値が非負のときには正しく動作することが多い

提出ファイル一覧

- ex4_1.pdf
- ex4_2_2.c
- ex4_2_4.c
- ex4_2_5.c(発展的課題)
- ex4_2_7.pdf(発展的課題)
- ex4_3.c

実行時間の測り方（一例）

- Windows powershell を用いる方法
<https://pig-log.com/windows-measure-time/>
- Mac（またはLinux）の time コマンドを用いる方法
<https://xtech.nikkei.com/it/atcl/column/15/042000103/080600050/?rt=nocnt>
- 明示的に禁止されている場合をのぞき、C言語標準規格に含まれる機能（bsearch, qsort など）や、使っている計算機の機能を活用して、課題作成に必要な時間と労力を節約して構いません。
- 自分でプログラムを書いて実行時間を測る方法は次ページ

プログラムにおける時間計測方法(再)

- ・ 実行ファイルの時間計測 ・ Mac, Linuxなどの time 命令
ターミナル (shell)で
% time a.out

- ・ プログラム中での時間計測 (基礎 Ex31解答例参照)

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

```
int main(void)
```

```
{
```

```
    clock_t start, end;  ← clock_t は long int
    .....
```

```
    start = clock();
```

```
    ここに計測対象のプログラム部分を置く
```

```
    end = clock();
```

```
    printf("Elapsed %ld(cycle) %lf (sec) ￥n", end-start,
           (double) (end-start)/CLOCKS_PER_SEC);
```

```
}
```

*スライドを読みやすくするために文字のフォントは適宜変更しています。

計測対象の実行時間が
短い場合は計測できない
(ばらつきが大きい)ので、
複数回数実行して計測
(同じことを1000回とか)

プログラムにおける時間計測方法(再)

- ・プログラム中での時間計測 (繰り返しによる計測の例)

```
#include <stdio.h>
#include <time.h>
#define NITER 1000
.....
```

```
int main(void)
```

```
{
```

```
    clock_t start, end;
```

```
    int iter;
```

```
    .....
```

```
    start = clock();
```

```
    for (iter=0; iter<NITER; iter++) {
```

```
        ここに計測対象のプログラム部分を置く
```

```
    }
```


```
    end = clock();
```

```
    printf("Elapsed %ld(cycle) %lf (sec) ￥n", end-start,
           (double) (end-start)/NITER/CLOCKS_PER_SEC);
```

```
}
```

* スライドを読みやすくするために文字のフォントは適宜変更しています。

標準関数qsortの紹介

- ソートはよく知られたアルゴリズムだから標準関数として用意して欲しい
- 要素の大きさや比較基準が変わっても、ソートの方法は本質的に変わらない。いちいち書き直すのは面倒くさい
- 標準関数としてクイックソートは用意されています

qsortの紹介 2

qsort(ポインタ, 要素数, 各要素のサイズ, 比較基準)
は配列を与えられた基準に従ってクイックソートする

ポインタ: 配列の先頭を指し示すポインタ

要素数: size_t型

各要素のサイズ: size_t型

比較基準: 比較を行う関数の名前を渡す。引数の型は「関数へのポインタ（?）」

qsort,bsearchの比較基準

比較基準は

int function(ポインタ1, ポインタ2)

のような、二つのポインタを引数にとり整数型を返す関数にする。ポインタ1で示される要素がポインタ2で示される要素より大きいとき正整数, 等しければゼロ, それ以外の場合負の整数を返すように関数を用意する
(例は次ページ)

qsortの使用例


```
#define N 20000
#define NUM_RANGE 10000

float my_random(float lower, float upper){省略}
void set_data(float a[], int n, int seed){省略}

int my_compare(const float *num1, const float *num2)
{ if (*num1>*num2) return 1; else if (*num1==*num2) return 0; else return
-1; }
// 整数を返す必要があるため return *num1-*num2 とすると浮動小数点が四捨五入されて
// 異なる *num1と*num2 に対して整数0が返されソートがうまくいかない

int main(void)
{
    float data[N];
    set_data(data, N, 1001);
    // sizeof data は配列dataの全要素を格納するために必要な容量を教える
    qsort(data, sizeof data / sizeof data[0], sizeof data[0], my_compare);
    printf("first: %f %f %f¥n", (double)data[0], (double)data[1],
(double)data[2]);
    printf("last: %f %f %f¥n", (double)data[N-3], (double)data[N-2],
(double)data[N-1]);
    return 0;
}
```

標準関数bsearchの紹介

- 探索はよく知られたアルゴリズムだから標準関数として用意して欲しい
- 要素の大きさや比較基準が変わっても、探索の方法は本質的に変わらない。いちいち書き直すのは面倒くさい
- 標準関数として二分探索は用意されています

二分探索bsearchの紹介 2

bsearch(ポインタ1,ポインタ2, 要素数, 各要素のサイズ, 比較基準)
は配列から指定された要素を二分探索で探す

ポインタ1: 探したい要素へのポインタ

ポインタ2: 配列の先頭を指し示すポインタ

要素数: size_t型

各要素のサイズ: size_t型

比較基準: 比較を行う関数の名前を渡す。引数の型は「関数へのポインタ（?）」

配列中の見つけた要素へのポインタを返す, 見つからないときはNULLを返す

bsearchの使用例

```
int data[100] = {0, 1, 2, 4, 5, 6, 7, 9, 10, 12, 13, 15, 16, 17, 18, 19, 20, 21,
23, 26, 27, 28, 29, 30, 31, 33, 34, 35, 37, 38, 39, 40, 41, 42, 44, 46,
47, 48, 49, 51, 52, 53, 54, 56, 57, 59, 60, 61, 63, 64, 65, 67, 68,
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 83, 84, 85, 86, 87, 89, 90, 91,
92, 93, 94, 95, 96, 97, 98, 99, 101, 102, 104, 105, 107, 108, 110,
111, 112, 113, 114, 115, 117, 118, 119, 120, 123, 124, 125, 127};

int my_compare(const int *num1, const int *num2) { return *num1 - *num2; }

int main(void)
{
    int key;
    int *place; // keyが見つかったポインタ
    printf("Input key: ");
    scanf("%d", &key);
    // sizeof data は配列データの全要素を格納するために必要な容量を教える
    place = bsearch(&key, data, sizeof data / sizeof data[0], sizeof data[0], my_compare);
    if (place == NULL) printf("見つかりませんでした💩\n");
    else {
        ptrdiff_t index = place - data; // ポインター同士の引き算の結果の型はptrdiff_t型になる
        // 上記の引き算の結果として *place == data[index] が成り立つようにindexの値が決まる
        printf("添字 %td に見つけた. data[%td] = %d\n", index, index, data[index]);
        // ptrdiff_t を printfするためには %td を用いる
    }
    return 0;
}
```

必要なヘッダーファイル

- ptrdiff_t 型は stddef.h で本日説明された typedef を用いて定義される
- bsearch, qsort は stdlib.h で定義される
- 必要な標準型や標準関数を用いる前に定義をincludeすることが望ましい

作る前に標準関数を探せ

- ベテランが作成した標準関数があるときにそれを自分で再度作るのは「車輪の再発明」に過ぎず、練習としてやる価値は大いにあるが、実践では避けるべきである
- C言語標準関数にはかなり色々な機能が用意されている
- C++ 標準関数は非常に豊富で、この後の課題のほぼすべてが標準関数にある（この科目でC++を提出すれば当然減点）

言語の習得とは

- その言語が提供する標準関数のほぼすべてを知り、自在に使いこなせることを含む
- 文法のすべてを知ること必要だがそれだけではない
- 従って、責任を伴うプログラムを作成するときには、使用する言語の標準関数のすべてを最初から終わりまで通読することが望ましい (C++ならできるがPythonやJavaは標準関数が多すぎる...)

うまく動かない質問時のお願い

- プログラムが上手く動作していないと疑わしい部分の変数を `printf` で表示するようにしてから質問して下さい（課題提出時にはデバッグ用`printf`は消す）
- （講師が松本のときのみ）質問する前に「`check1.sh`」によるコンパイラの警告と実行時エラーを無くして下さい。警告やエラーの意味がわからないときは気軽に聞いてください

コピペレポートについて

プログラムや考察などが他の提出者と重複している場合、不正とみなして減点および問い合わせをすることがあります