Practice of Information Processing

（IMACU）

Seventh lecture, part 1
Previous exercises

Makoto Hirota

Previous Exercise 6-3 Swap values swap.c

- Create a program swap.c that inputs two integers and displays them in a different order.
  - Define the swap function according to the main function on the right
  - The swap function takes a pointer as an argument like a prototype declaration

```c
#include <stdio.h>
/* prototype declaration */
void swap(int *x, int *y);

int main(void){
    int a,b;

    /* Input 2 integers */
    scanf("%d" &a);

    /* Exchange the contents */
    swap(&a, &b);

    /* Display results */
    printf("%d %d¥n", a, b);

    return 0;
}
```

Display example

```
10 20 (enter)
20 10
```

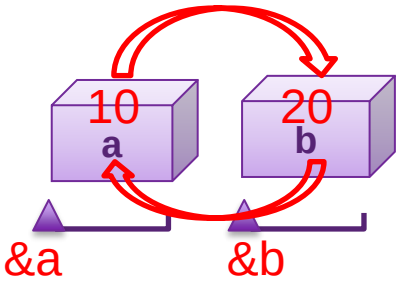
Previous Exercise 6-3: swap.c Example


```
int main(void)
{
    int a, b

    scanf("%d %d", &a, &b)
    swap(&a, &b)
    printf("%d %d\n", a, b)

    return 0
}
```

This function can swap the contents of two variables without returning anything as a return value

```
void swap(int *px, int *py)
{
    int tmp

    tmp = *px
    *px = *py
    *py = tmp
}
```

The two values at the pointers can be changed in the function.



Exercise 6-6

- Create a program ‘pi_count.c’ which can count the number of “0,” “1,” “2,” “3,” “4,” “5,” “6,” “7,” “8,” “9” in the “pi.txt”. “pi.txt” can be downloaded from Google classroom.

```
int main(void){
int c;
FILE* fp;
int count[10]={0,0,0,0,0,0,0,0,0,0};
.
.
while((c=fgetc(fp))!=EOF){
  if( c == '0' ) count[0]=count[0]+1;
  else if(c =='1' ) count[1]=count[1]+1;
  else if(c =='2' ) count[2]=count[2]+1;
   .
   .
   .
}
```

Exercise 6-6: Model answer


```
int main(void){
  int c,i;
  FILE* fp;
  int count[10]={0,0,0,0,0,0,0,0,0,0};

  fp=fopen("pi.txt","r");
  while((c=fgetc(fp))!=EOF){
    if( c == '0' ) count[0]=count[0]+1;
    else if(c =='1' ) count[1]=count[1]+1;
    else if(c =='2' ) count[2]=count[2]+1;
    .
    .
    else if(c =='9' ) count[9]=count[9]+1;
  }
  fclose(fp);

  for(i=0;i<10;i++)
     printf("# of %d is %d¥n",i,count[i]);
}
```

Output:

```
# of 0 is 1954
# of 1 is 1997
# of 2 is 1986
# of 3 is 1987
# of 4 is 2043
# of 5 is 2082
# of 6 is 2017
# of 7 is 1953
# of 8 is 1962
# of 9 is 2020
```

Exercise 6-7

- Create the following program ‘addone.c’.
  - Read a text file ‘input.txt’ that contains one line:

    Input Parameters: 5, 7, 3

  - and store 5, 7 and 3 into the variables `a`, `b` and `c` , respectively.
  - `a`, `b` and `c` are defined as shown at right. You must use `pc` instead of `c`.
  - Write a function, `addone`, that increases `a`, `b` and `c` by one.
  - Namely, the output should be

    6 8 4

```
#include <stdio.h>

int a;

/* define addone function here */
void addone( ???? ){
     ????
}

int main(void){
     FILE *fp;
     int b, c;
     int *pc; pc=&c;
     /* Don't use c below*/

     /* Read input.txt */

     addone( ???? );
     printf("%d %d %d¥n",a,b,*pc);
     return 0;
}
```

Exercise 6-7: Model answer

- fscanf is useful in many cases.
- In principle, all variables can be made global. Then, the source code becomes difficult to read. Global variables should be used as little as possible.

```
#include <stdio.h>

int a;
     The same variable
void addone(int *pb,int *pc){ /* a shouldn't be here */
     a+=1;
     *pb+=1;
     *pc+=1;
}
                                The same name, but different
int main(void){
     FILE *fp;
     int b, c;
     int *pc; pc=&c;

     fp = fopen("input.txt","r");
     if(fp==NULL){ printf("Can't find file¥n"); return 1; }

     fscanf(fp,"Input Parameters: %d,%d,%d", &a,&b,pc);
     fclose(fp);

     addone(&b,pc);
     printf("%d %d %d¥n",a,b,*pc);
     return 0;
}
```

Practice of Information Processing

（IMACU）

Seventh lecture part 1：Pointer（2）

Makoto Hirota

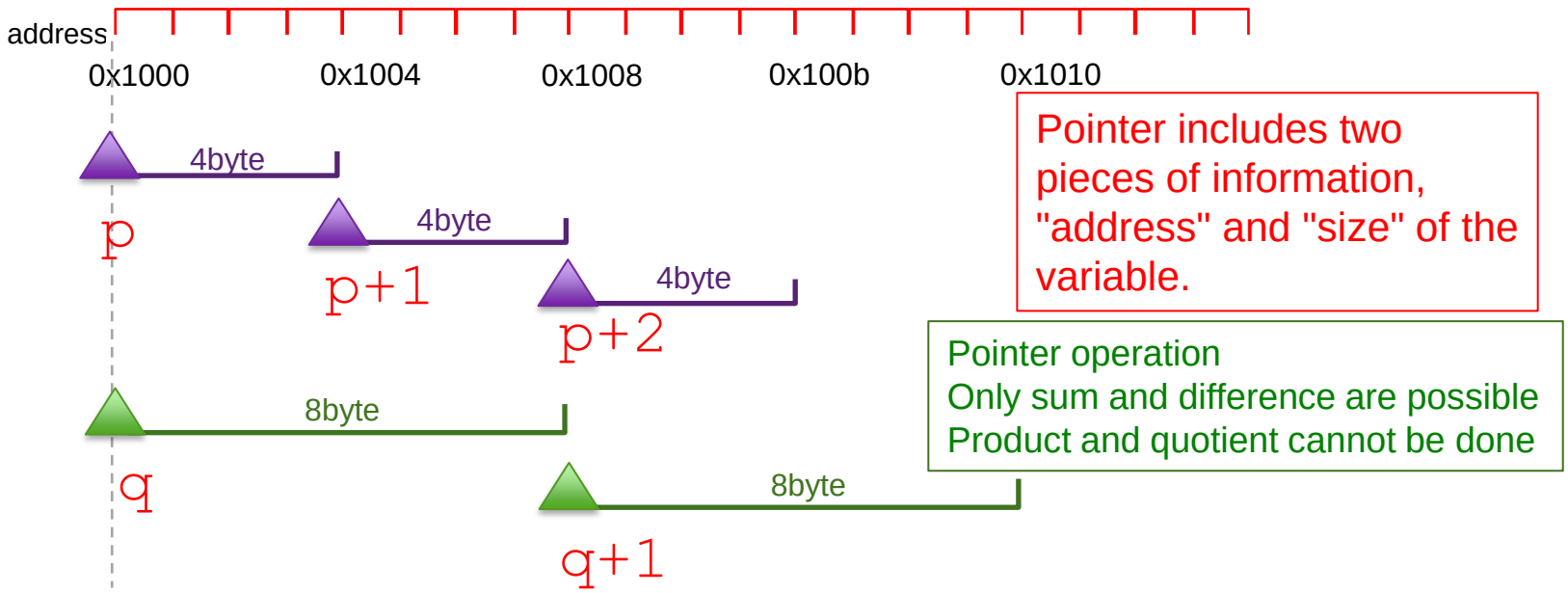
Contents of this lecture

- Pointer
  - Pointer to array
  - Pointer to structure
- Final assignment

Pointer operation

- By adding (or subtracting) the integer value `n` to the pointer, the address can be moved by (the size of the type) x n.

（例）
```
int *p
double *q
```
（int type pointer: 4byte）
（double type pointer：8byte）

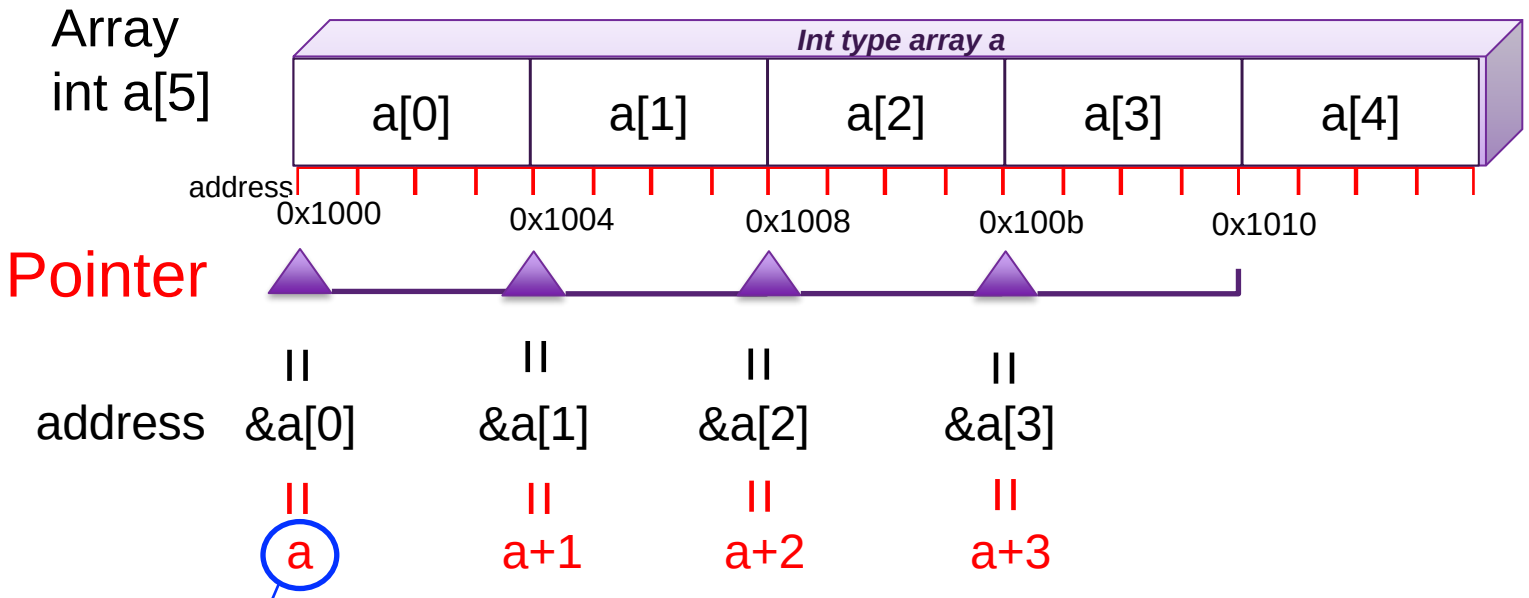


Relationship between arrays and pointers

- Elements of array can be also accessed by pointer.
  - Array `a[i]` and pointer reference `*(a+i)` have the same meaning.
  - The array name `a` can also be used as a pointer.

`a[i] = *(a+i)`

Equivalent!



【important】

That is, the array name a is equivalent to a pointer to the start address of the array.

* In practice, `a[0], a[1], …` are easier to read than `*(a), *(a + 1), …`

Exercise 7-1 Array reference by pointer operation

- For the int type array a [5] = {10,20,30,40,50}, refer to the contents of the array a [] by using the separately defined int type pointer p, and perform pointer operation. Create a program array_pointer.c that displays the contents of an array

  Display example:

```
a[0] = 10
a[1] = 20
a[2] = 30
a[3] = 40
a[4] = 50
```

Exercise: 7-1 array_pointer.c tips


```
#include <stdio.h>

int main(){

    /* variable declaration */
    int a[5]={10,20,30,40,50};
    int *p; /* pointer */
    int i;

    /* processing contents */
    p = a; /* assign start
              address into pointer */
    for(i=0; i < 5; i++){
        printf("a[%d] ¥n",i,      );
    }

    return 0;
}
```

Don't use a but p.

Start address of array (= Pointer to array) How to get

The next two have the same meaning

```
p = a;
```

The array name is a pointer to the first array element

```
p = &a[0];
```

Substitute the address of the first array element

Exercise 7-2 Program maxmin.c that returns two results

- Complete the following program maxmin.c by implementing the function maxmin_array () that returns the maximum and minimum values of the five input integer arrays.

```c
#include <stdio.h>

int main(){
    int max,min;
    int array[LEN];
    int i;

    /* assaign array */
    for(i=0; i < LEN; i++){
        printf("array[%d] = ",i);
        scanf("%d",&array[i]);
    }

    /* find max-min value from array */
    maxmin_array(array, &max, &min);

    printf("max:%d min:%d¥n",max, min);

    return 0;
}
```

Implementing function

```c
/* prototype declaration */
void maxmin_array(int array[], int *pmax, int *pmin);
```

* The first argument, `int array []`, specifies that array is an array name. In other words, a pointer to an int type `"int *"` may be used, but it is written as `array []` to make it easier to understand that the first argument is a pointer to an array.

* Variable names can be omitted in the prototype declaration.
  `void maxmin_array (int [], int *, int *);`
However, it is better to leave the variable name because it makes it easier to understand the meaning.

Exercise7-2 maxmin.c Example


```
int main()
{
    int max, min;
    int array[LEN];
    int i;

    for(i=0; i<LEN; i++){
        printf("array[%d] = ",i);
        scanf("%d", &array[i]);
    }
    maxmin_array(array, &max, &min);
    printf("max:%d min:%d\n",max, min);

    return 0;
}
```

min

max

Keep Memory

Pass the memory address with &

```
void maxmin_array(int array[], int *pmax, int *pmin)
{
    int i;

    *pmax=array[0];
    *pmin=array[0];

    for(i=1; i<LEN; i++){
        if(*pmax < array[i])
            *pmax = array[i];
        if(*pmin > array[i])
            *pmin = array[i];
    }
}
```

int type array array
(Variable name array is equivalent to pointer)

Pointer to int type pmax, pmin

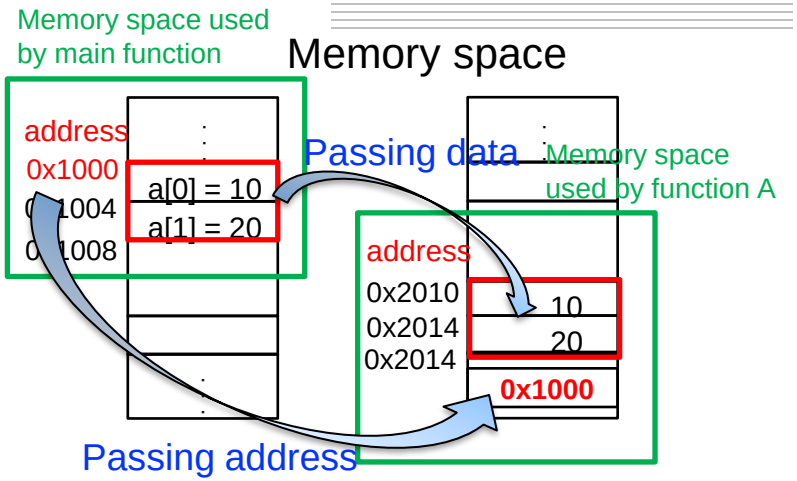
← Substitute the value of array [0] as the initial value of pmax and pmin. Since pmax and pmin are pointers, how to manipulate the values?

The answer obtained by the function maxmin_array is substituted to min, max in the memory area of the main function, via the pointers pmax and pmin.

Summary: Two ways to exchange data among functions

Example: Call function A from main function

- An array, a[0] and a[1], is defined in main function.
- I want to pass the values of a[0] and a[1] to function A.



(1) Passing data = “call by value"

Pass the contents (= values) of a [0] and a [1] to function A.

```
funcA(a[0],a[1])
```

(Copy the values to new local variables of function A.)

Good! The function is forbidden to change the original variable
Bad When the array size is large, it takes extra memory area and processing time to copy.

(2) Pass the address = “call by reference"

Pass only the first address of the array (“pointer”) to function A.

```
funcA(a)
```

Function A refers to all values by the address and, moreover, can overwrite them.

Good! Any large data can be passed quickly without consuming extra memory space.
Good! The function can modify the values of the original array.
Bad The original array might be broken by function A.
Bad Require extreme caution to the size of array.

Previous lecture: Structure

- Defined as a new type (= structure) to handle multiple different variables with one name

We can refer the member (component) by "Variable_name.(dot)member_name"

Defined outside the main function at the beginning of the program

```
struct person {
        char name[100];
        double height;
        double weight;
} ;
```
←semicolon ";" is required

Can be declared as a variable type in a program

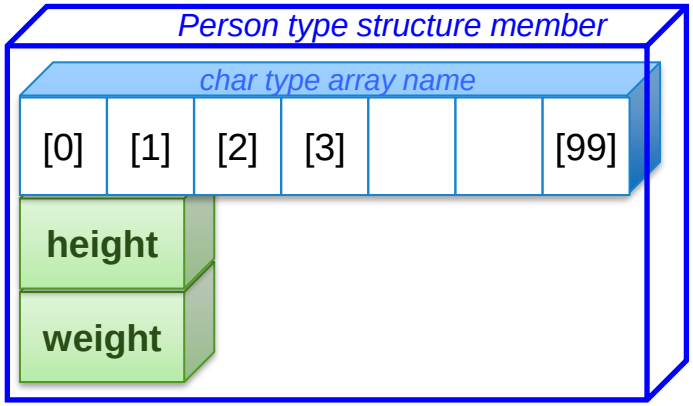
```
struct person member1, member2;
```
(Declare 2 variables)

struct person: Type name

member1, member2: variable variable

```
member1.height = 170.5;
member1.weight = 62.0;
```
(first person)

```
member2.height = 165.0;
member2.weight = 55.3;
```
(second person)



Person type structure

struct person

Note that this combination is the name of the type

You can also give the type an alias for simplicity

```
typedef struct person st_person;
```

alias name

Pointer to structure

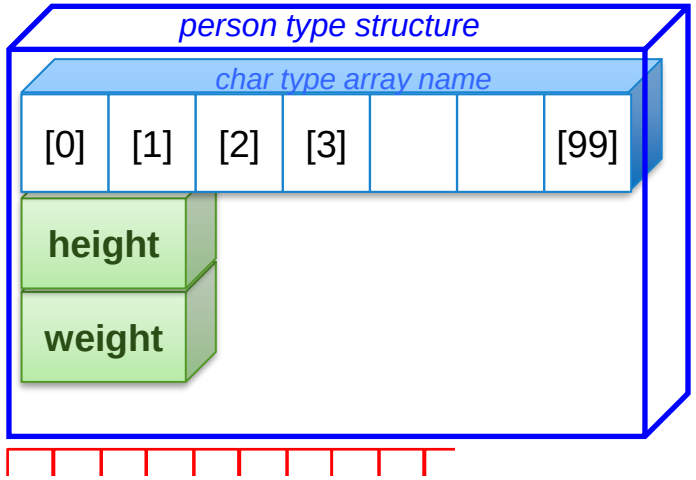
- Declaring a pointer to a structure

```
struct structure_name * structure_pointer variable_name;
```

(Just add * in front of the variable name like other data types)

```
struct person *p_member;

st_person *p_member;
```

※ * Here, for the sake of clarity, the variable name "p_**"
is used to clearly indicate that the structure variable is a pointer.



Up to this point, it is the same as a normal variable type pointer.

The pointer variable "p_member" points to the start address of the person type variable.
* The size is for the entire structure

Member reference via pointer to structure

- If you try to refer to the contents of the pointer (member of the structure) using "*" as before, you need to do as follows.

```
struct person  *p_member;

(*p_member).height = 170.5;
(*p_member).weight = 62.0;
```

Since the member reference operator "." Of the structure has a higher precedence than "*", it cannot be simply written as "* member.height".

- To make it easier to describe, special symbols (arrow operator "->") are provided only for pointers to structures.

Pointer_of_structure->name_of_member

```
struct person  *p_member;

p_member->height = 170.5;
p_member->weight = 62.0;
```

Summary of member reference method of structure

When the structure is a normal variable: member.height

When the structure is represented by a pointer: p_member ->height

Exercise:7-3 Structure pointer struct_pointer.c

- Modify the sample program struct_person.c (shown below) into a program struct_pointer.c that similarly displays the contents of the structure by replacing the boxed lines with a function print_person () .
- The argument of the function print_person () must be one pointer variable of the st_person type.

＜struct_person.c＞

```
int main(void){
    /* variable declaration */
    st_person member[MEMBER_NUM] =
        {{"Ichiro",160,50},{"Jiro",170,60},{"Saburo",180,70}};

    int i;

    /* processing contents */
    for(i=0; i < MEMBER_NUM; i++){
        printf("Name=%s¥n",member[i].name);
        printf("  Height=%.2f¥n",member[i].height);
        printf("  Weight=%.2f¥n",member[i].weight);
    }
    return 0;
}
```

Replace with the function print_person ()

Supplement：Exercise 7-3 struct_pointer.c

- main function

```
int main(void){
    /* variable declaration */
    st_person member[MEMBER_NUM] =
        {{"Ichiro",160,50},{"Jiro",170,60},{"Saburo",180,70}};

    int i;

    /* processing contents */
    for(i=0; i < MEMBER_NUM; i++){
        print_person(&member[i]);
    }
    return 0;
}
```

- Tips: print_person function

```
void print_person(st_person *person)
{
    /* Process */
}
```

Exercise 7-4: pointer to structure struct_pointer2.c

Submission required

- Modify the function print_person () of struct_pointer.c to a new function print_person_id() which receives “a pointer to the st_person type array” and “an ID number” (array subscript).
- Then, create a program struct_pointer2.c that displays the contents of the structure in the following way.
  - In the main function, enter the ID from the keyboard to display the contents of the corresponding structure.
  - See what happens if you enter an ID that is equal to or larger than MEMBER_NUM.
  - If an invalid ID is entered, perform error handling.

Supplement: Exercise 7-4 struct_pointer2.c

- Main function

```
int main(void){
    /* variable declaration */
    st_person member[MEMBER_NUM] =
        {{"Ichiro",160,50},{"Jiro",170,60},{"Saburo",180,70}};

    int i;

    /* processing contents */
    printf("ID? ");
    scanf("%d? ", &id);
    printf("¥n");

    if(id >= 0 && id < MEMBER_NUM) {
        print_person_id(member,id);
    }else{
        exit(1);
    }
    return 0;
}
```

Send a pointer to the beginning of the array and an ID to the function

`member` can be `&member[0]`

Causion:

#Include <stdlib.h> is required when using the exit () function for error handling

- Tips: print_person_id function

```
void print_person_id(st_person person[], int id)
{
    /* process */
}
```

Equivalent to `st_person *person`

Here, in order to clarify that the argument is a pointer to the "array", the argument is intentionally declared as **`st_person person []`** in an array style.
It has the same meaning as **`st_person *person`**

Dynamic Memory Allocation

- So far, array size was hard-coded by macro definition.
- We can allocate an arbitrary size of array during the execution of a program.
  - In addition to #include <stdio.h>, #include <stdlib.h> is required
  - Memory allocation
    - **`p=malloc(b);`**
      - b bytes are reserved on memory, and the first address p is returned.
      - Be careful not to give too large value of b.
    - The first address p can be treated as array.

```
int *p;
p=malloc(3*sizeof(int));
```

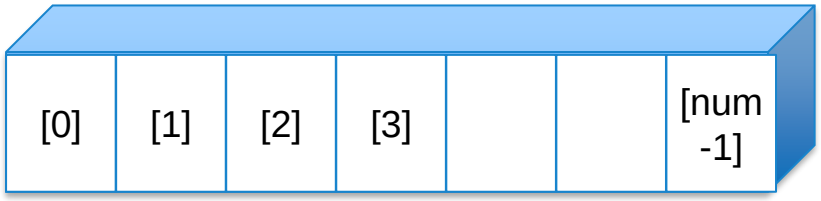
      → Integer array p[0], p[1], p[2] is allocated.
  - Memory release
    - **`free(p)`**;
      - release the reserved memory
      - If you don't write this, the memory is reserved until the end of execution.

Exercise 7.5: Practice of memory allocation

Submission required

- Compile and execute the following program, malloc_test.c. Then change the number (num) of input data. Observe what happens when num is smaller or larger than 10.

```
#include <stdio.h>
#include <stdlib.h>
Int main()
{
      int num,i;
      int array[10];
      int* ptr;

/**** scan the data from keyboard****/
      printf("Number of Data? ");
      scanf("%d",&num);

/**** read and display the data****/
/* array version*/
      for(i=0;i<num;i++){
          scanf("%d",&array[i]);
      }

/* malloc version*/
      ptr = malloc(sizeof(int)*num);
      if(ptr==NULL){

/* the case of the memory does
                  not have enough space*/
      printf("memory allocation error¥n");
      return 1;
      }
```



Array size can be specified by a variable.

```
      for(i=0;i<num;i++){
            scanf("%d",&ptr[i]);
      }

/* Display the input data*/
      for(i=0;i<num;i++){
          printf("%d %d¥n",array[i],ptr[i]);
      }

/* Display the memory usage*/
      printf("¥n--- memory usage ---¥n");
      printf("array : %d [bytes]¥n", sizeof(array));
      printf("malloc: %d [bytes]¥n", num*sizeof(int));

/* Release the reserved memory space*/
      free(ptr);
      return 0;
}
```

Practice of Information Processing

（IMACU）

Seventh lecture Part 2：Final assignment

Makoto Hirota

Contents of this lecture

- Pointer
  - Pointer to array
  - Pointer to structure
- Final assignment

Final assignment

Refer to “Final Assignment” in the Google Classroom.

Tips：Reading the CSV file

- Run the sample program read_csv_sample.c
- J_dataset.csv is a comma-separated text file

Example of CSV file

```
Antlers,18,9,7,54,30
Bellmare,10,6,18,40,63
Cerezo,18,5,11,39,25
Consadole,13,7,14,54,49
F.C.Tokyo,19,7,8,46,29
...
```

```
sscanf(buff,"%[^,],%d,%d,%d,%d,%d",
        name, &win, &draw,
        &loss, &GF, &GA);
```

When trying to match with the format `"%s,%d,%d,%d,%d,%d"`, the delimiters `','` (comma) are also read as a character string.

Therefore, use the format `"%[^,]"`
Since `'^'` means “other than that”, `[^,]` matches other than commas. As a result, the first string will be assigned to `name` until a comma appears.

Tips: Difference between the two templates

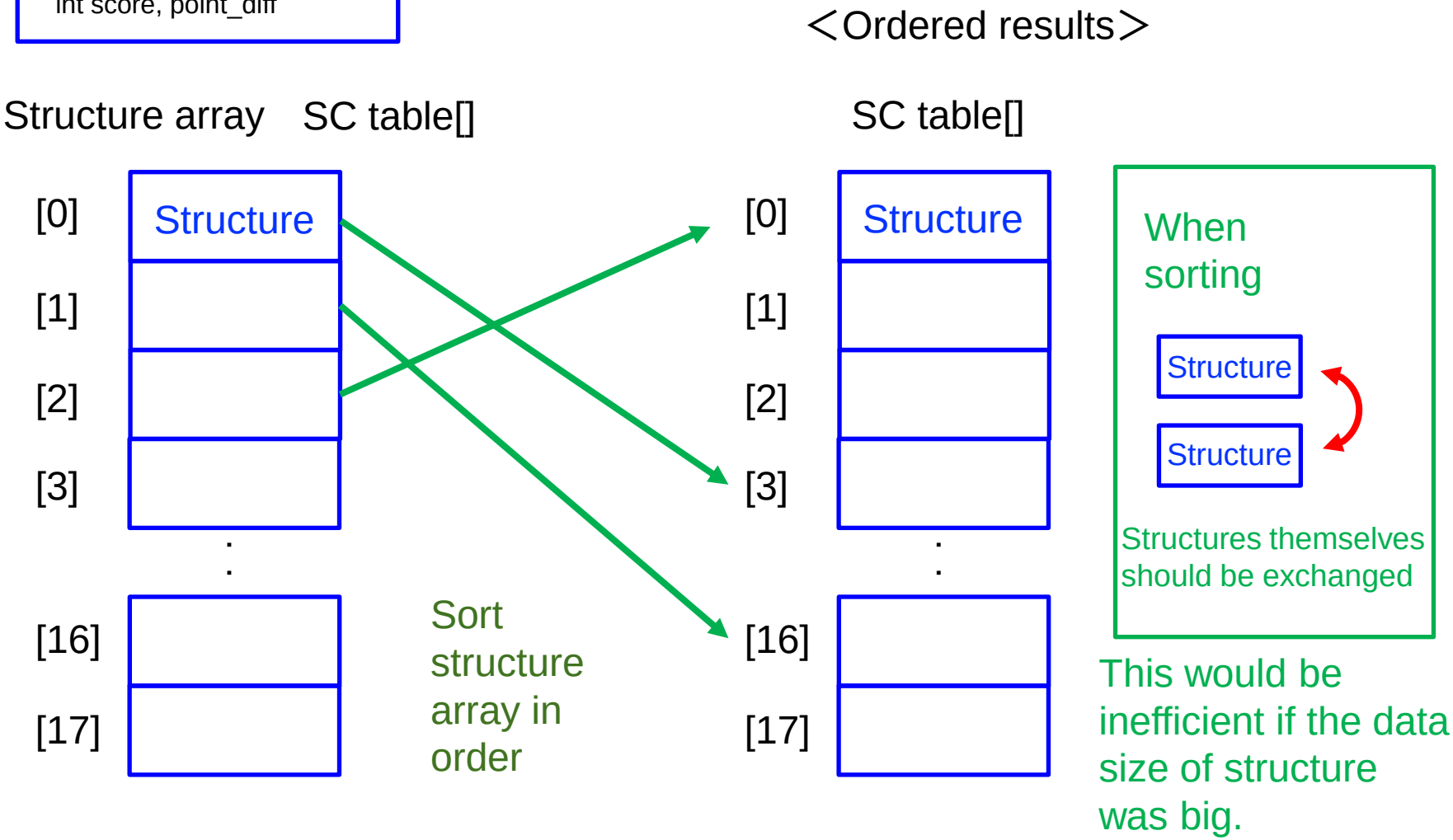
- There are two templates:
  J_score_template1.c and J_score_template2.c
- Both handle the data of all teams in the structure array `table[]`.

```c
/*team score structure */
struct team_score{
    char name[NAME_LENGTH]; /* team name */
    int win;  /* number of win */
    int draw; /* number of draw */
    int loss; /* number of lose */
    int GF;   /* total goals scored */
    int GA;   /* total goals conceded */
    int score; /* win point */
    int point_diff; /* goals difference */
};

/*  alias */
typedef struct team_score SC;
```

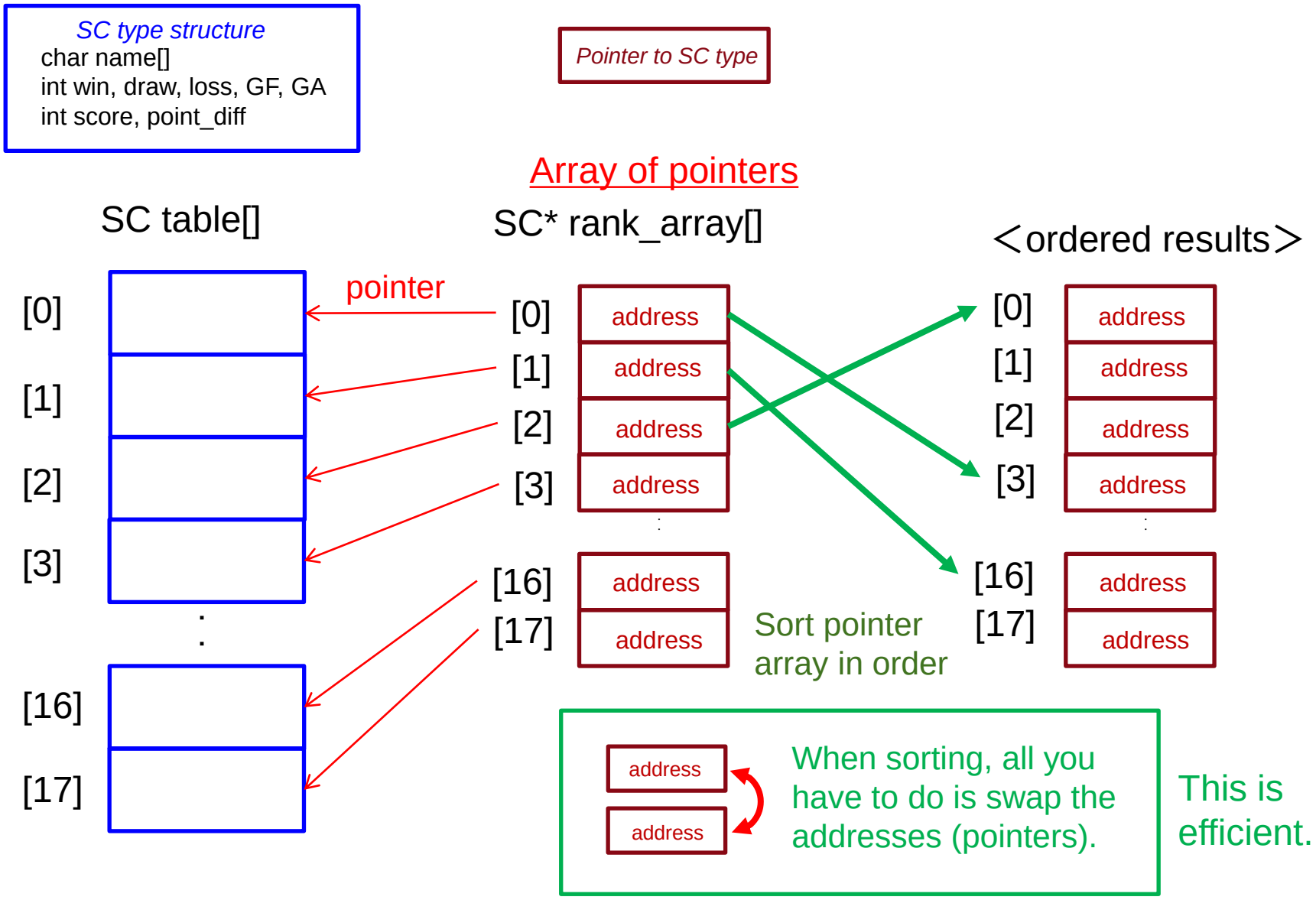
- J_score_template1.c :
  Sort the structure array directly by using bubble sort or selection sort.
- J_score_template2.c :
  - Keep the order of the structure array unchanged.
  - Sort the array of pointers to the structure array without directly sorting the structure array.

Schematic of templete1

SC type structure
char name[]
int win, draw, loss, GF, GA
int score, point_diff



Schematic of templete2



Tips: Remark on pointers in function arguments

- There is no difference between `SC *team` and `SC team[]` in function arguments (`team` is pointer to `SC`).
- However, make it easy to distinguish whether the argument is a pointer to a structure or a structure array.

```
void read_data(FILE *fin, SC table[]);
void calc_score(SC *team);
void rank_score(SC table[]);
void swap_SC(SC *team1, SC *team2);
void write_data(FILE *fout, SC table[]);
```

`table[]` is structure array

`team` is pointer to structure

All the arguments of these functions are "pointers"

`SC *team` → `team` is a pointer to a variable of SC structure type (data of one team, `*team`)

`SC table[]` → `table` is a pointer to the beginning of an array of SC structure types (data of all teams, `table[0], table[1],…`)

The reader can recognize whether the pointer is array or not.

Final announcement

- All the classes are finished!
- Good luck.