Programming Basics (HW#3)

Data Structure

Im Y. Jung

Problem

Get the family names, ages, Math scores, English scores, History scores of the students in a class through a file.

Save them using “struct" and "dynamic memory allocation".

And, print out their sorted results based on the field a user chooses.

The requirements are as follows.

- Because you can not know the number of students in a class in advance,

you should use “dynamic memory allocation” of “struct”.

- You should check whether the ages are the integers greater than 0,

and the scores are in [0, 100] during input process.

- You should sort the family names(alphabet order), the ages and the scores in ascending order.

The data with the same rank may be not ordered.

If a user choose a field to sort by, the sorted results of the students should be printed out by the field.

Problem

- Execution

```
1) Insert
2) Sort
3) Quit
Select a menu : 2
Result : There is no data to be sorted. Program terminates.

1) Insert
2) Sort
3) Quit
Select a menu : 1
File name : input1.txt
Result : The age cannot be a negative number. Program terminates.
```

<Test file>

```
[input1.txt]
kim -2 11 10 100
lee 9 18 90 55
```

Problem

<Test file>

```
[input2.txt]
gu 10 12 50 99
kim 24 20 50 34
lee 23 33 40 33
sung 30 40 22 12
```

```
1) Insert
2) Sort
3) Quit
Select a menu : 1
File name : input2.txt
Result :
No Name Age Math English History
1 gu 10 12 50 99
2 kim 24 20 50 34
3 lee 23 33 40 33
4 sung 30 40 22 12
```

```
1) Insert
2) Sort
3) Quit
Select a menu : 2
1) Name
2) Age
3) Math
4) English
5) History
Choose the field to sort by : 4
No English Name Age Math History
1 22 sung 30 40 12
2 40 lee 23 33 33
3 50 gu 10 12 99
4 50 kim 24 20 34
```

```
1) Insert
2) Sort
3) Quit
Select a menu : 2
1) Name
2) Age
3) Math
4) English
5) History
Choose the field to sort by : 1
No Name Age Math English History
1 gu 10 12 50 99
2 kim 24 20 50 34
3 lee 23 33 40 33
4 sung 30 40 22 12
```

```
1) Insert
2) Sort
3) Quit
Select a menu : 3
```

Example – Problem Analysis (1)

- File Input : name, age, Math, English, History
- Keyboard Input : insert, sort, quit
- Output :
  - **Insert : get data from the file given**
  - **Sort : print out the sorted results**
    - Name : alphabet order
    - Age, score : ascending order
  - **Quit : program terminates**

- What to do
  - **In/Out Design**
    - File in/Keyboard in/Screen out
  - **Get the data from File, store it, and check the inputs**
  - **Get the user’s choice, process, and print out**

- What to use
  - **Data/storage Design**
    - Data type designed using struct

Example – Problem Analysis (2)

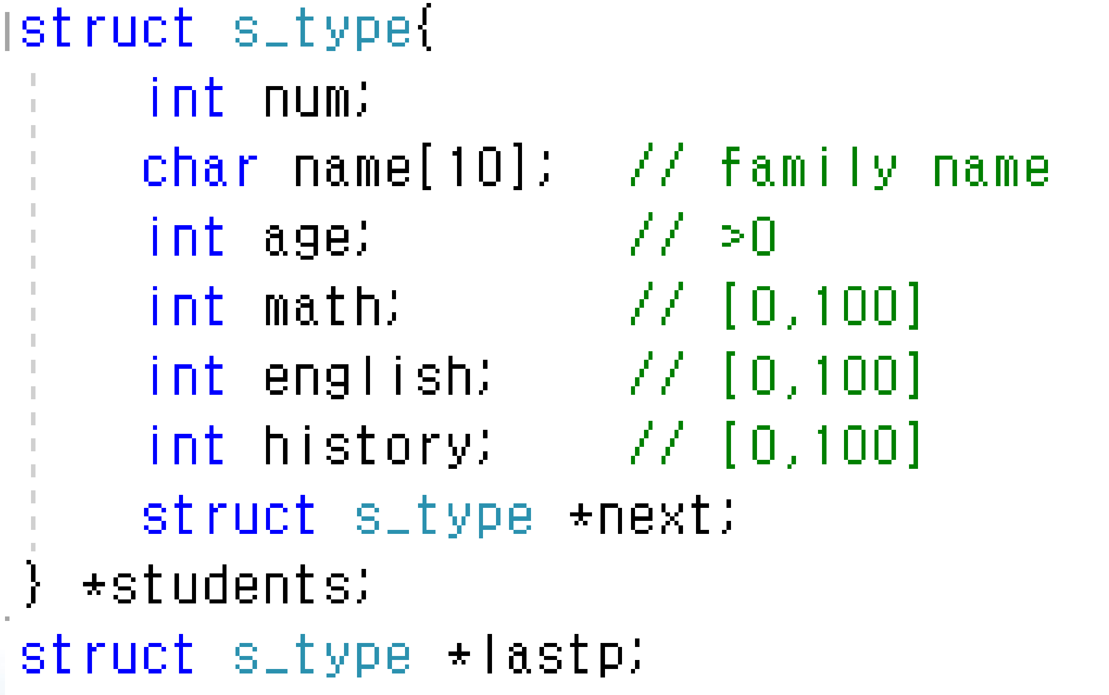
- Requirements :
  - **Use “dynamic memory allocation” of “struct”**
  - **Check whether the ages are the integers greater than 0, and the scores are in [0, 100] during input process.**
  - **If a user choose a field to sort by, the sorted results of the students’ data should be printed out by the field.**

- What to use
  - **malloc()**

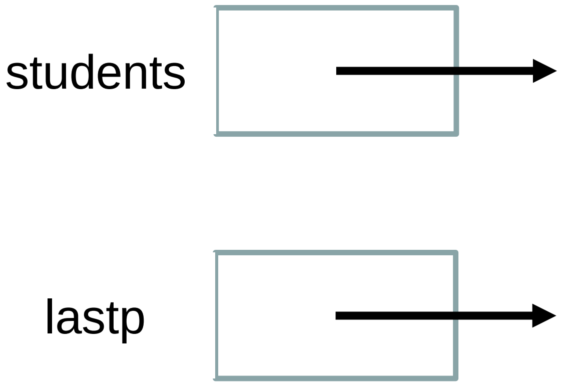
Example – Problem Analysis (3)

- How to do
  - **Program structure**
    - Several functions ?
      - Get data from a file, allocate a space from memory and store the data to the memory
      - Sort the data according to the user's choice, Print out the sorted data
      - Free the allocated memory
  - **Algorithm**
    - Get the user's choice
      - Insert :
        - » Get the file name
        - » Request of memory allocation
        - » Get student's data, check the data, and store it
      - Sort :
        - » Get the field to sort by
        - » Sort the data by the field and print out
      - Quit :
        - » Free the allocated memory
        - » Program terminates

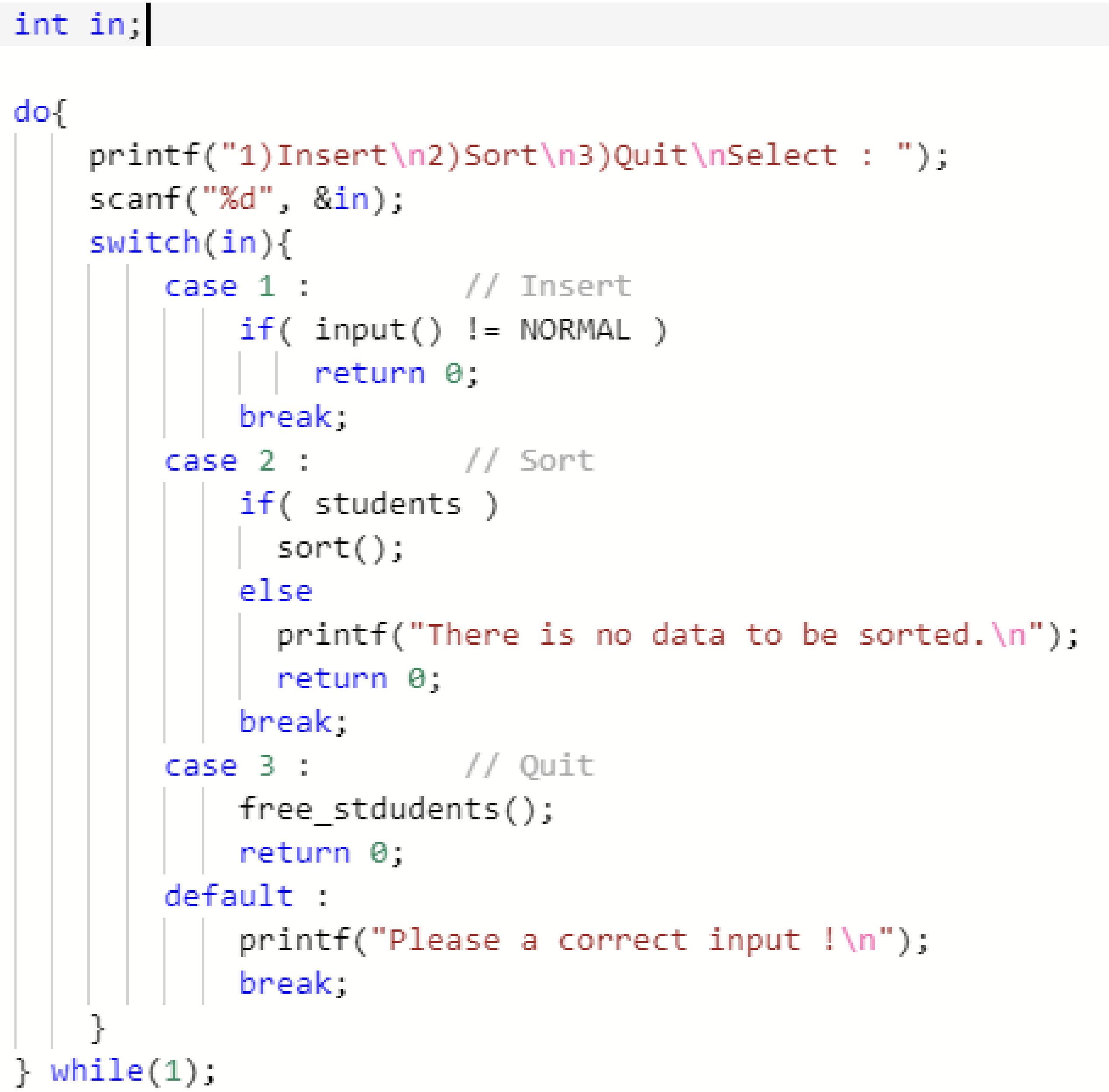
Example – Data/storage Design



```
struct s_type{
    int num;
    char name[10];   // family name
    int age;         // >0
    int math;        // [0,100]
    int english;     // [0,100]
    int history;     // [0,100]
    struct s_type *next;
} *students;
struct s_type *lastp;
```



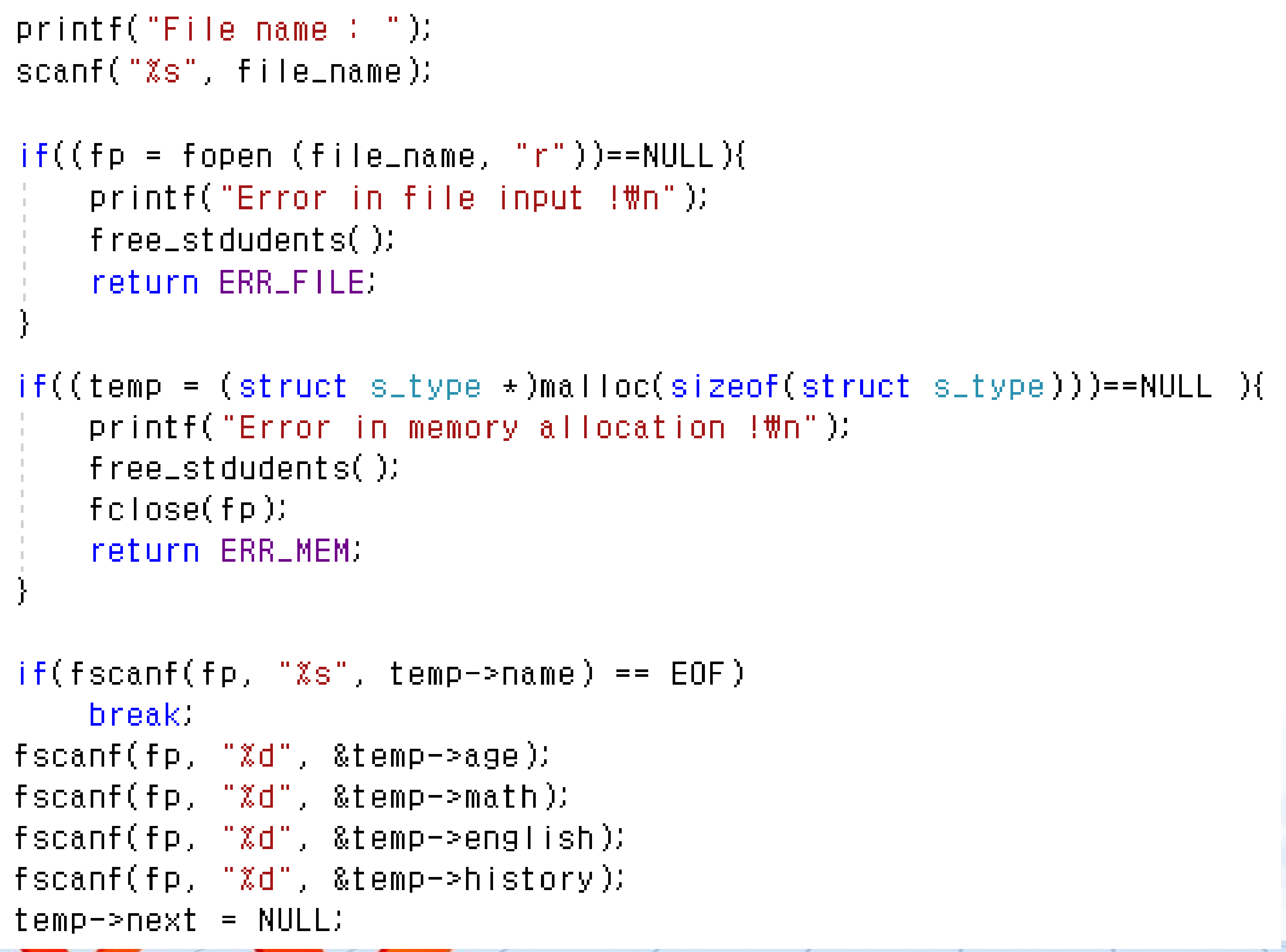
Example – Program Flow (1)



```
int in;

do{
    printf("1)Insert\n2)Sort\n3)Quit\nSelect : ");
    scanf("%d", &in);
    switch(in){
        case 1 :        // Insert
            if( input() != NORMAL )
                return 0;
            break;
        case 2 :        // Sort
            if( students )
              sort();
            else
              printf("There is no data to be sorted.\n");
              return 0;
            break;
        case 3 :        // Quit
            free_stdudents();
            return 0;
        default :
            printf("Please a correct input !\n");
            break;
    }
} while(1);
```

- Get a user’s choice
  - **Insert :**
    - Call input()
  - **Sort :**
    - Call sort()
  - **Quit :**
    - Call free_students()
    - Program terminates

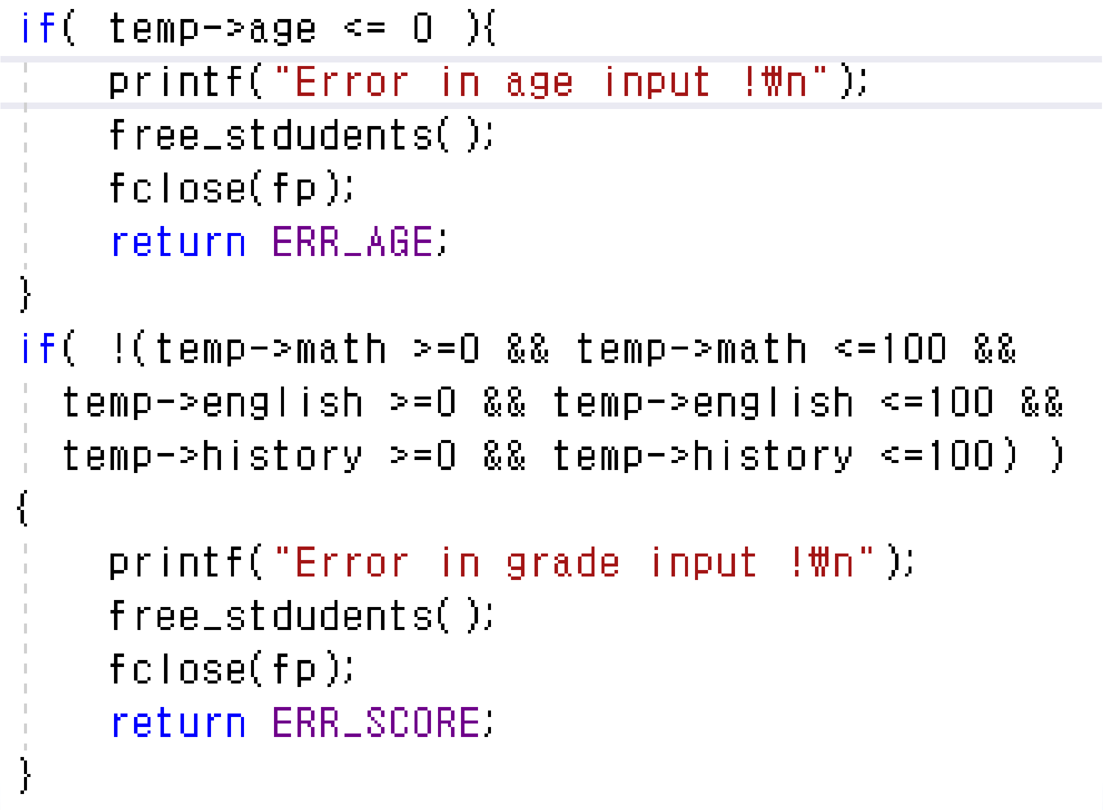
Example – Program Flow (2), input()



```
printf("File name : ");
scanf("%s", file_name);

if((fp = fopen (file_name, "r"))==NULL){
    printf("Error in file input !₩n");
    free_stdudents();
    return ERR_FILE;
}
if((temp = (struct s_type *)malloc(sizeof(struct s_type)))==NULL ){
    printf("Error in memory allocation !₩n");
    free_stdudents();
    fclose(fp);
    return ERR_MEM;
}

if(fscanf(fp, "%s", temp->name) == EOF)
    break;
fscanf(fp, "%d", &temp->age);
fscanf(fp, "%d", &temp->math);
fscanf(fp, "%d", &temp->english);
fscanf(fp, "%d", &temp->history);
temp->next = NULL;
```

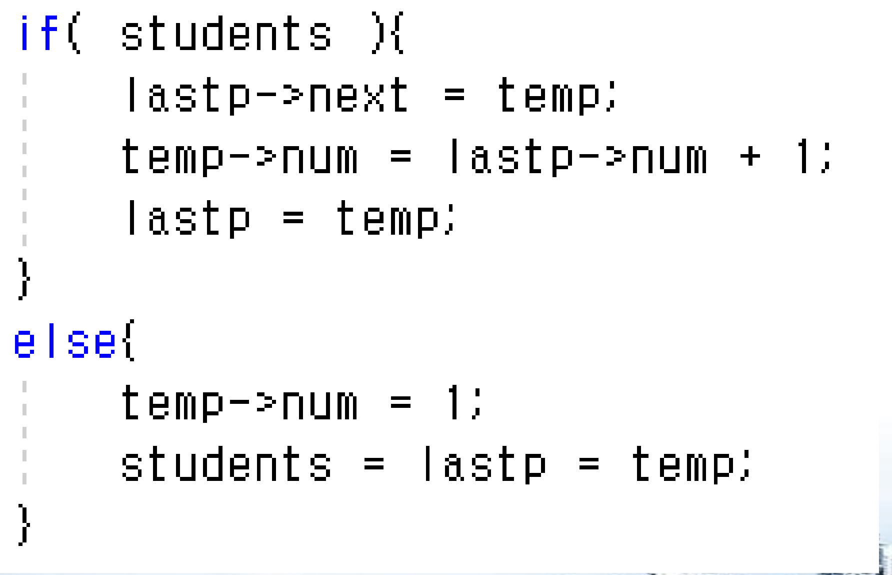
- File open
- Memory allocation for a data node
- Get data and store it to the data node (temp)

Example – Program Flow (3), input()



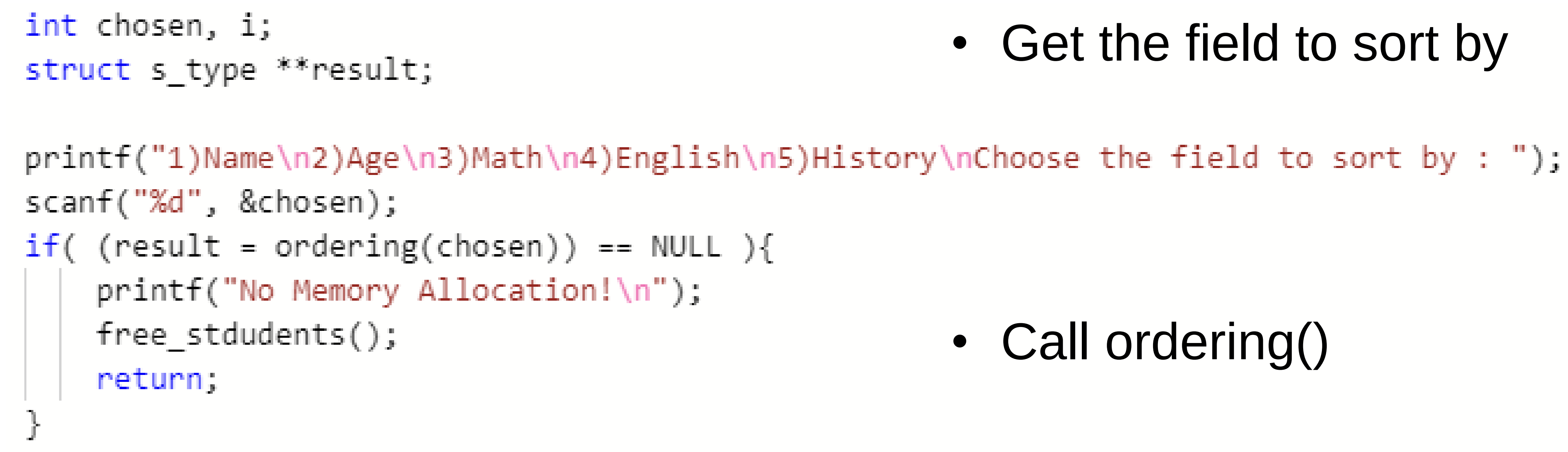
```
if( temp->age <= 0 ){
    printf("Error in age input !₩n");
    free_stdudents();
    fclose(fp);
    return ERR_AGE;
}
if( !(temp->math >=0 && temp->math <=100 &&
  temp->english >=0 && temp->english <=100 &&
  temp->history >=0 && temp->history <=100) )
{
    printf("Error in grade input !₩n");
    free_stdudents();
    fclose(fp);
    return ERR_SCORE;
}
```

- Check the data
- Insert the data node to students



```
if( students ){
    lastp->next = temp;
    temp->num = lastp->num + 1;
    lastp = temp;
}
else{
    temp->num = 1;
    students = lastp = temp;
}
```

Example – Program Flow (4), sort()



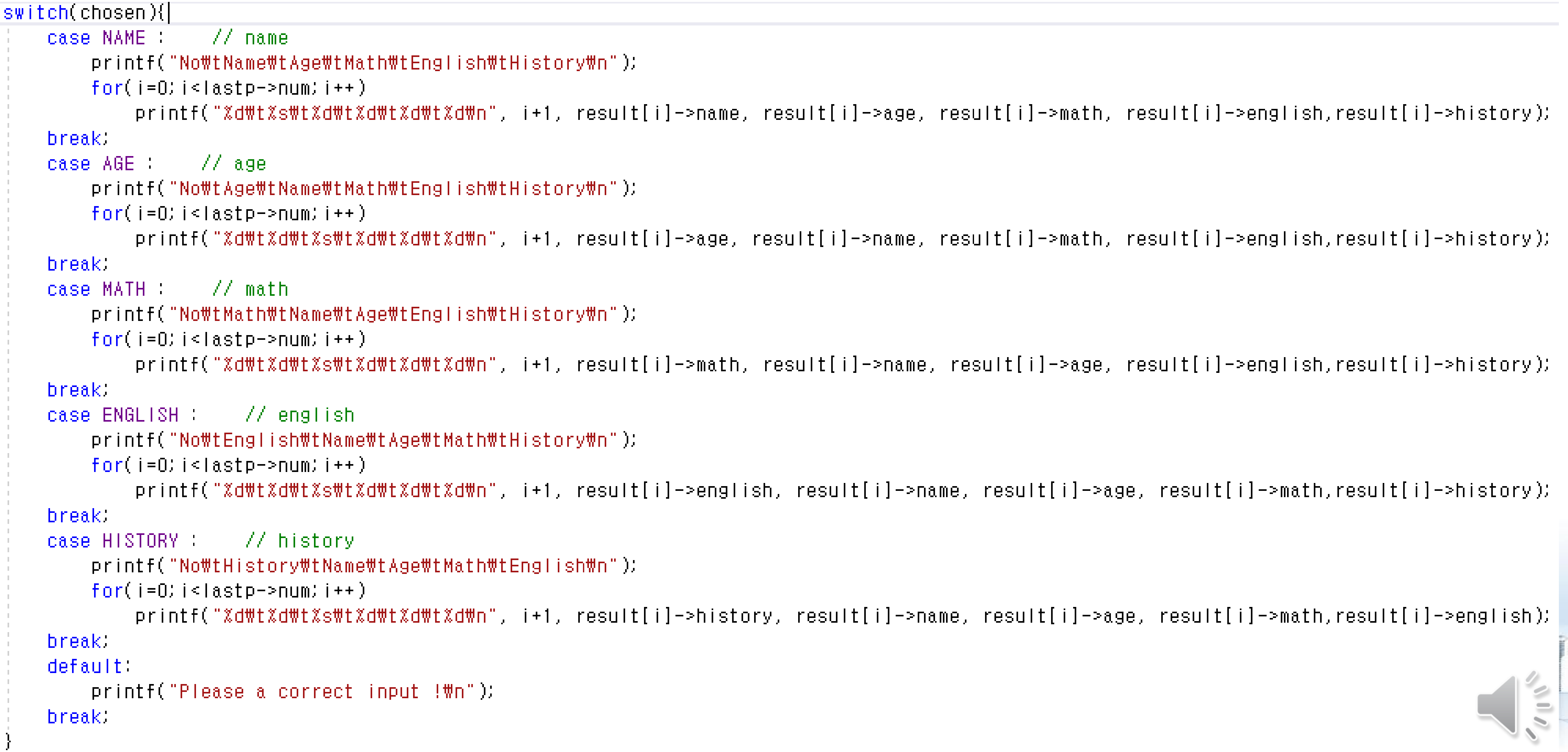
```
int chosen, i;
struct s_type **result;

printf("1)Name\n2)Age\n3)Math\n4)English\n5)History\nChoose the field to sort by : ");
scanf("%d", &chosen);
if( (result = ordering(chosen)) == NULL ){
    printf("No Memory Allocation!\n");
    free_stdudents();
    return;
}
```

- Get the field to sort by

- Call ordering()

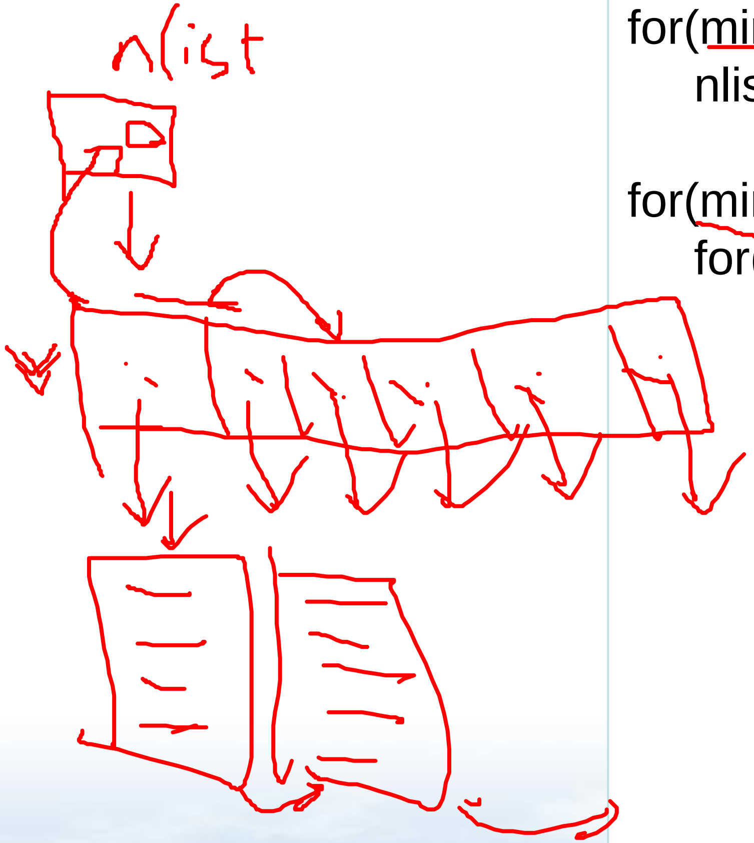
Example – Program Flow (5), sort()

- Print out the results



```
switch(chosen){
    case NAME :    // name
        printf("No₩tName₩tAge₩tMath₩tEnglish₩tHistory₩n");
        for( i=0; i<lastp->num; i++)
            printf("%d₩t%s₩t%d₩t%d₩t%d₩t%d₩n", i+1, result[i]->name, result[i]->age, result[i]->math, result[i]->english,result[i]->history);
    break;
    case AGE :    // age
        printf("No₩tAge₩tName₩tMath₩tEnglish₩tHistory₩n");
        for( i=0; i<lastp->num; i++)
            printf("%d₩t%d₩t%s₩t%d₩t%d₩t%d₩n", i+1, result[i]->age, result[i]->name, result[i]->math, result[i]->english,result[i]->history);
    break;
    case MATH :    // math
        printf("No₩tMath₩tName₩tAge₩tEnglish₩tHistory₩n");
        for( i=0; i<lastp->num; i++)
            printf("%d₩t%d₩t%s₩t%d₩t%d₩t%d₩n", i+1, result[i]->math, result[i]->name, result[i]->age, result[i]->english,result[i]->history);
    break;
    case ENGLISH :    // english
        printf("No₩tEnglish₩tName₩tAge₩tMath₩tHistory₩n");
        for( i=0; i<lastp->num; i++)
            printf("%d₩t%d₩t%s₩t%d₩t%d₩t%d₩n", i+1, result[i]->english, result[i]->name, result[i]->age, result[i]->math,result[i]->history);
    break;
    case HISTORY :    // history
        printf("No₩tHistory₩tName₩tAge₩tMath₩tEnglish₩n");
        for( i=0; i<lastp->num; i++)
            printf("%d₩t%d₩t%s₩t%d₩t%d₩t%d₩n", i+1, result[i]->history, result[i]->name, result[i]->age, result[i]->math,result[i]->english);
    break;
    default:
        printf("Please a correct input !₩n");
    break;
}
```

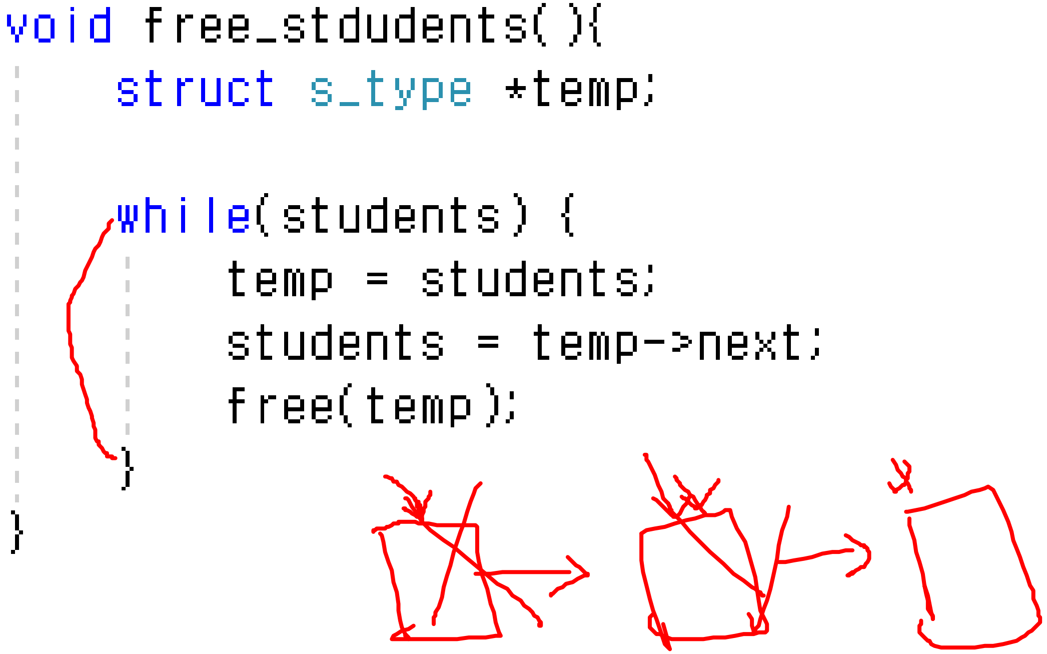
Example – Progra



```
struct s_type **nlist, *p, *temp;
int min, i ;

nlist = (struct s_type **)malloc(sizeof(struct s_type *)*(lastp->num));

for(min=0, p=students;p && min<lastp->num ; min++, p=p->next)
    nlist[min] = p;

for(min=0 ; min<lastp->num ; min++){
    for(i=min+1; i<lastp->num ; i++){
        switch(c){
            case NAME :
                if(strcmp(nlist[min]->name, nlist[i]->name)>0)
                {
                    temp = nlist[min];
                    nlist[min] = nlist[i];
                    nlist[i] = temp;
                }
            break;
        }
    }
}

return nlist;
```

- Sort the data nodes by the field chosen
- Store them to struct *nlist[]

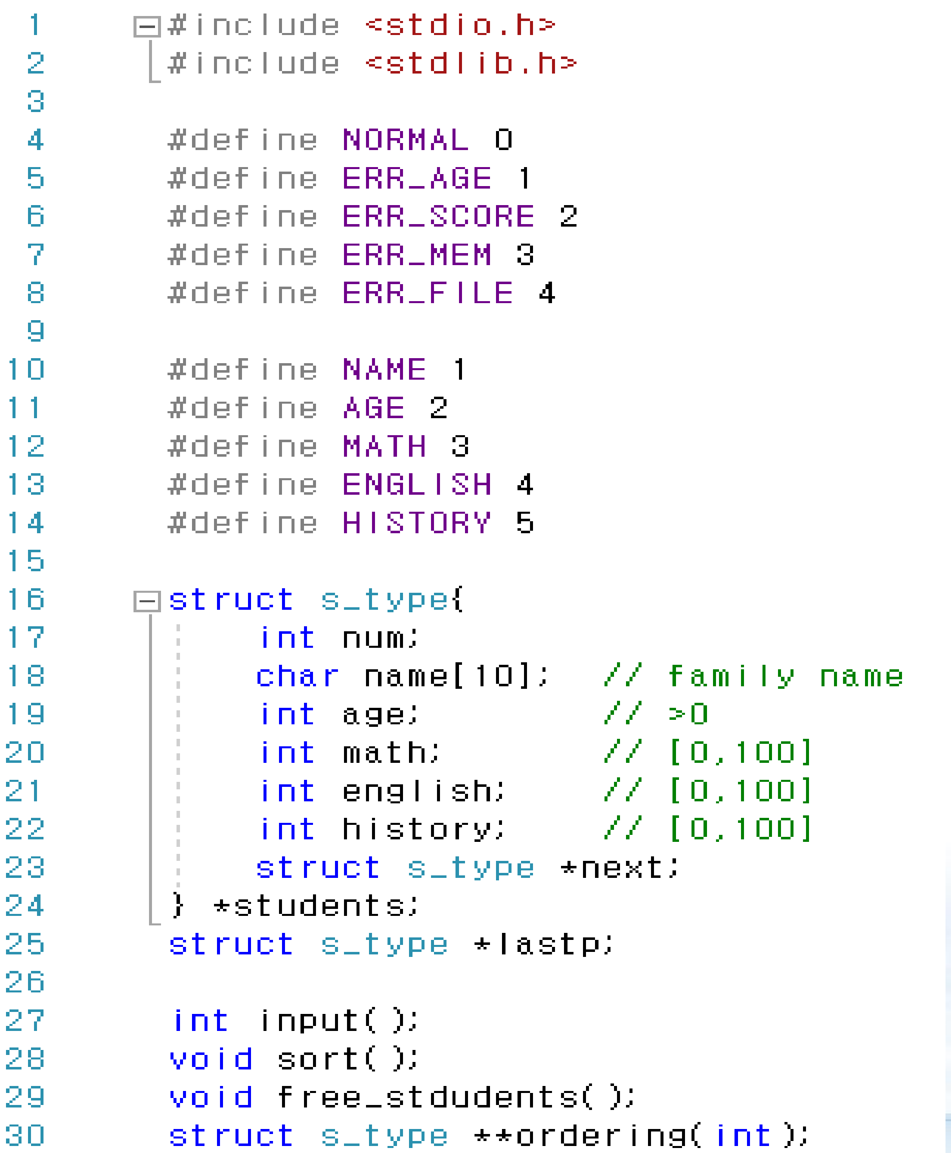
Example – Program Flow (7), free_students()



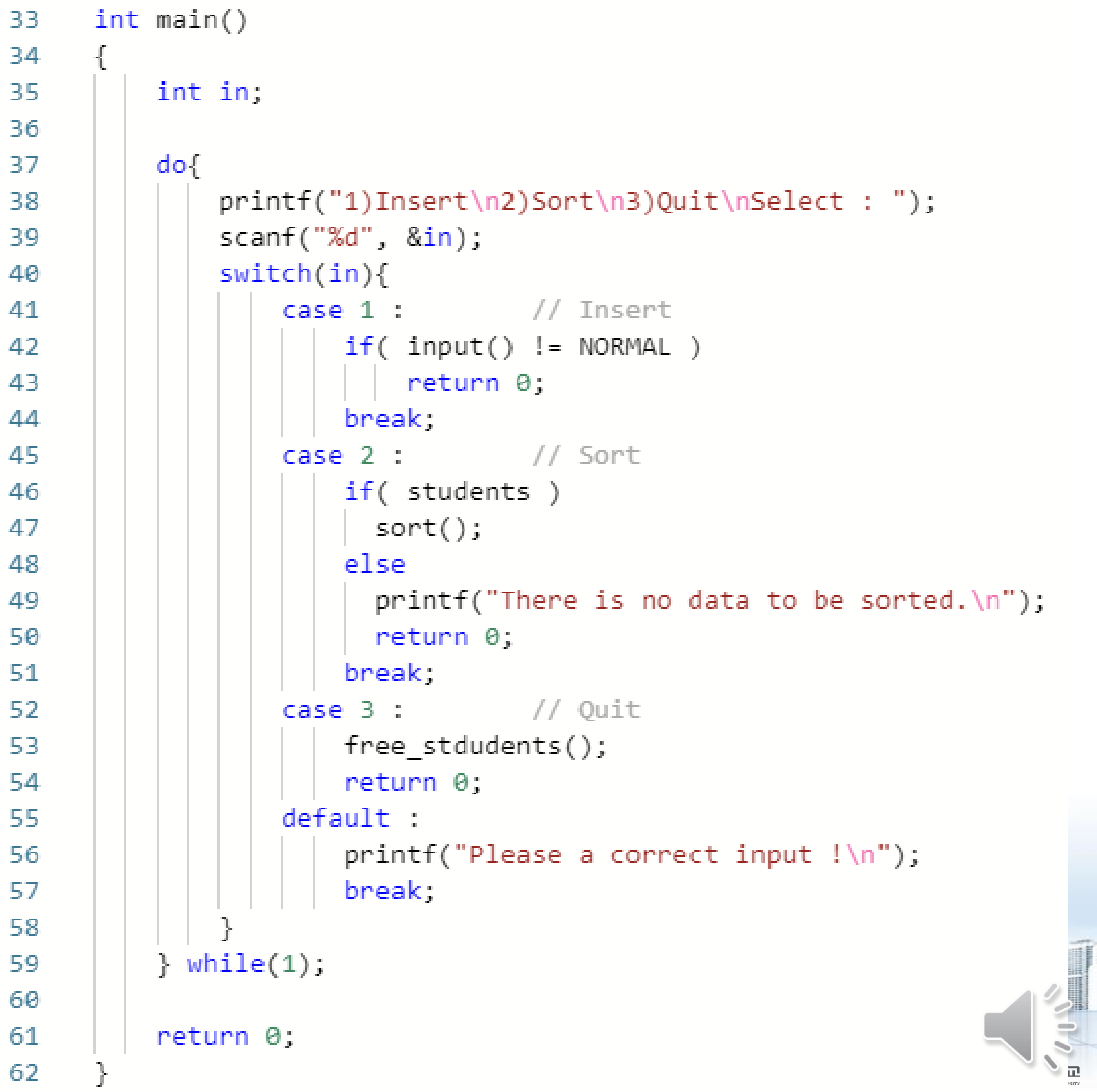
```
void free_stdudents( ){
    struct s_type *temp;

    while(students) {
        temp = students;
        students = temp->next;
        free(temp);
    }
}
```

- Free the data nodes

Sample (1/4)

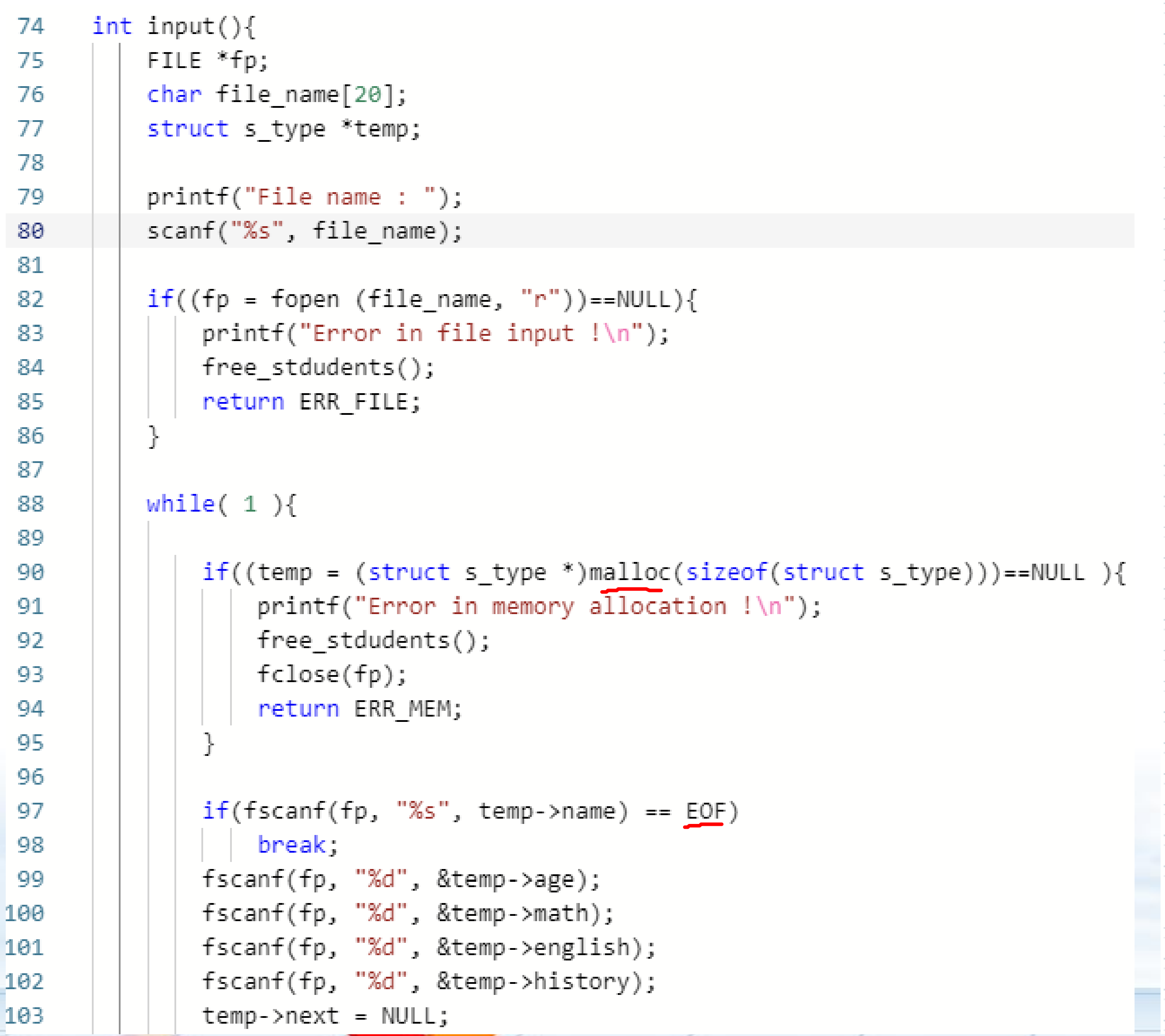


```
#include <stdio.h>
#include <stdlib.h>

#define NORMAL 0
#define ERR_AGE 1
#define ERR_SCORE 2
#define ERR_MEM 3
#define ERR_FILE 4

#define NAME 1
#define AGE 2
#define MATH 3
#define ENGLISH 4
#define HISTORY 5

struct s_type{
    int num;
    char name[10];  // family name
    int age;        // >0
    int math;       // [0,100]
    int english;    // [0,100]
    int history;    // [0,100]
    struct s_type *next;
} *students;
struct s_type *lastp;

int input();
void sort();
void free_stdudents();
struct s_type **ordering(int);
```

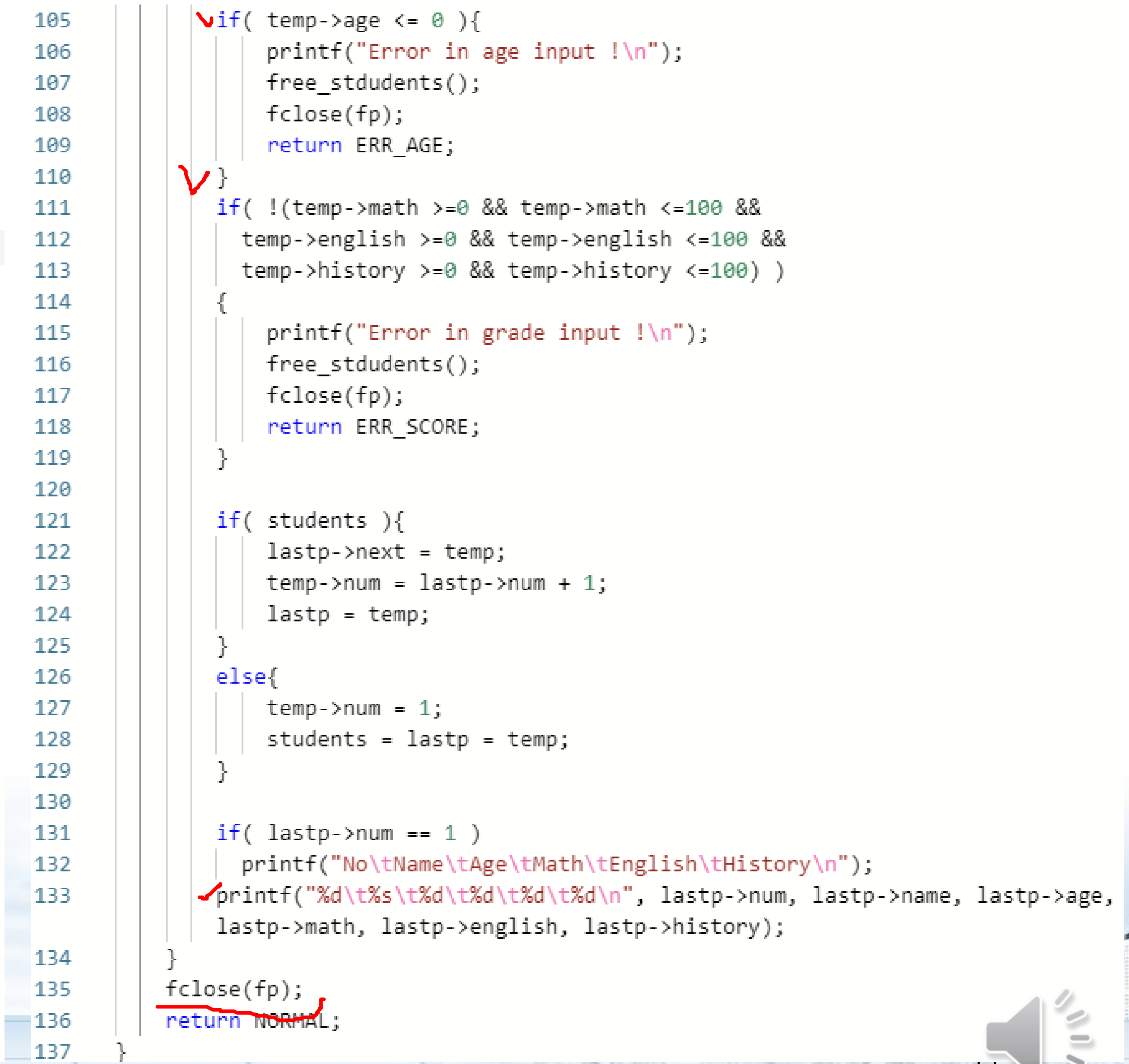


```
int main()
{
    int in;

    do{
        printf("1)Insert\n2)Sort\n3)Quit\nSelect : ");
        scanf("%d", &in);
        switch(in){
            case 1 :         // Insert
                if( input() != NORMAL )
                    return 0;
                break;
            case 2 :         // Sort
                if( students )
                    sort();
                else
                    printf("There is no data to be sorted.\n");
                    return 0;
                break;
            case 3 :         // Quit
                free_stdudents();
                return 0;
            default :
                printf("Please a correct input !\n");
                break;
        }
    } while(1);

    return 0;
}
```

Sample (2/4)

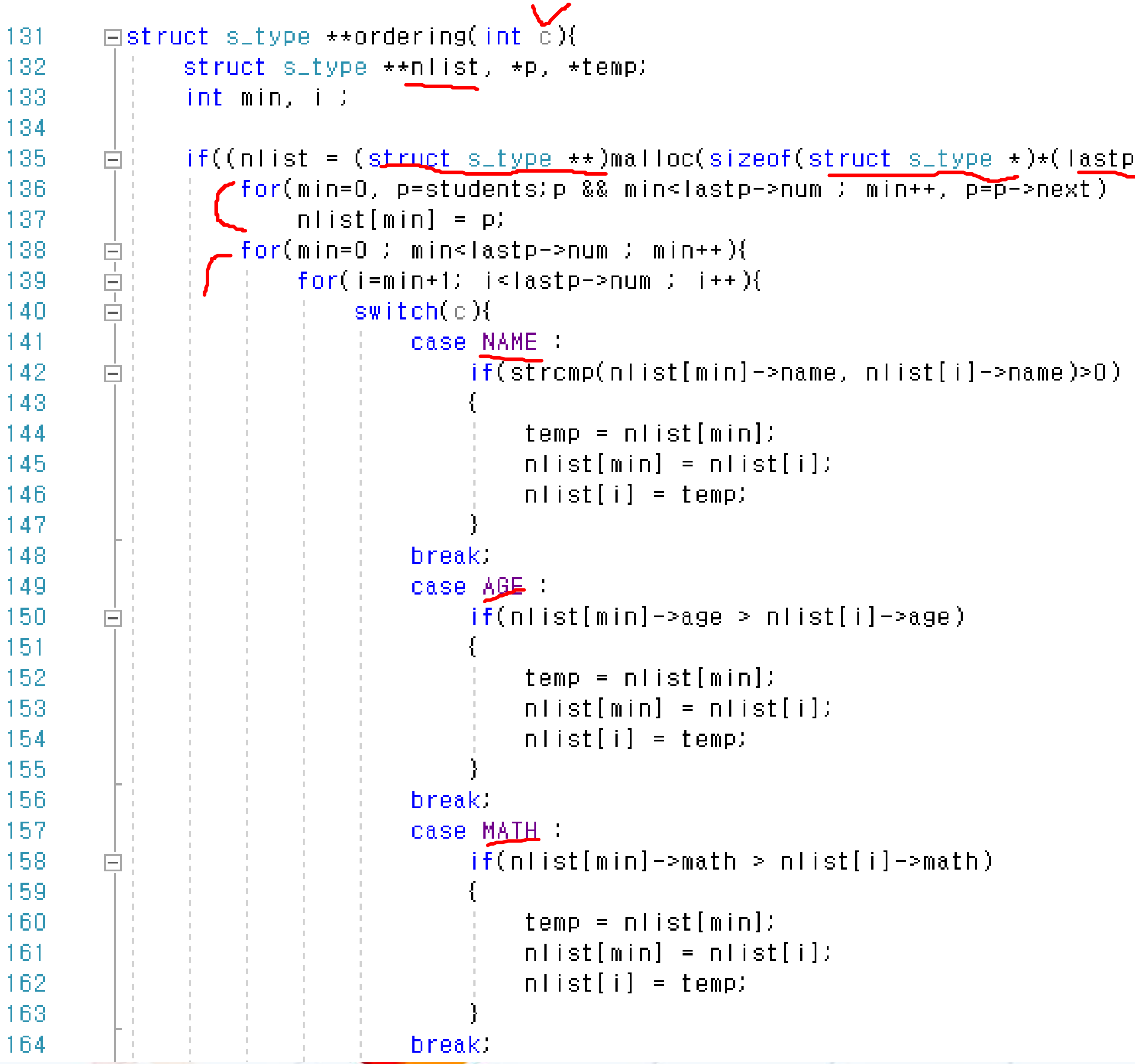


```
int input(){
    FILE *fp;
    char file_name[20];
    struct s_type *temp;

    printf("File name : ");
    scanf("%s", file_name);

    if((fp = fopen (file_name, "r"))==NULL){
        printf("Error in file input !\n");
        free_stdudents();
        return ERR_FILE;
    }

    while( 1 ){

        if((temp = (struct s_type *)malloc(sizeof(struct s_type)))==NULL ){
            printf("Error in memory allocation !\n");
            free_stdudents();
            fclose(fp);
            return ERR_MEM;
        }

        if(fscanf(fp, "%s", temp->name) == EOF)
            break;
        fscanf(fp, "%d", &temp->age);
        fscanf(fp, "%d", &temp->math);
        fscanf(fp, "%d", &temp->english);
        fscanf(fp, "%d", &temp->history);
        temp->next = NULL;
```

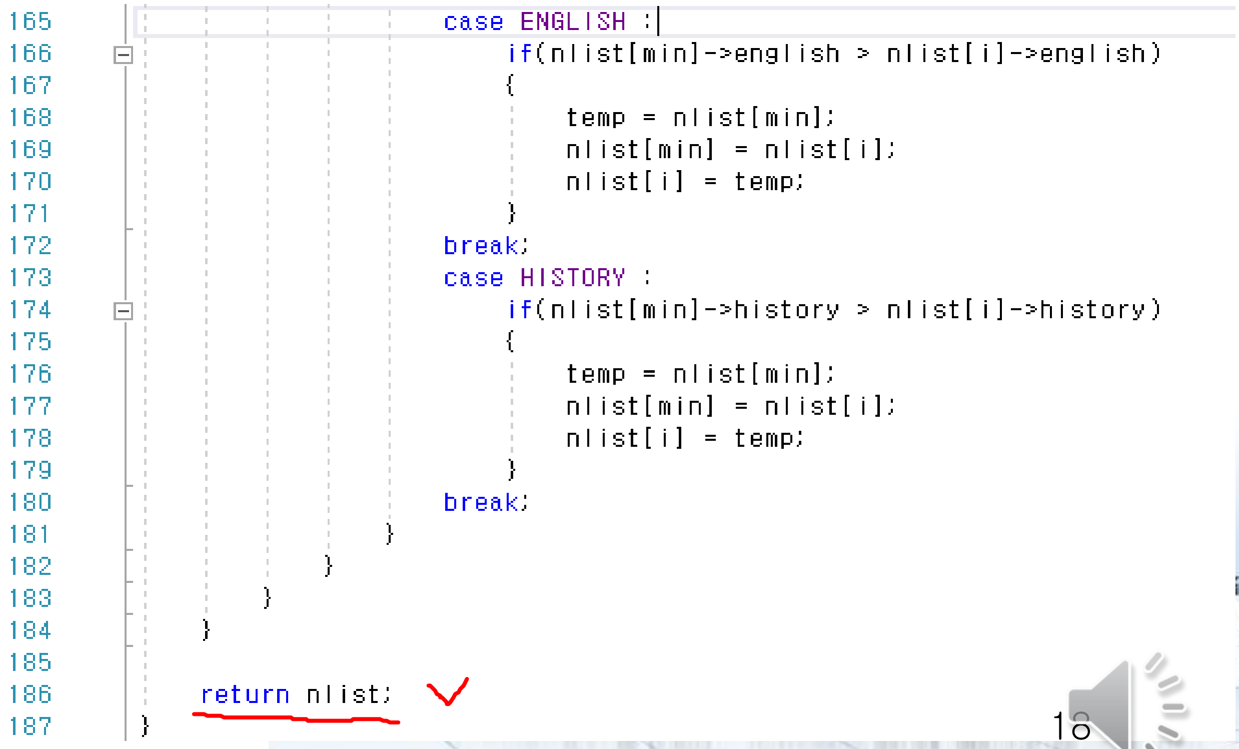


```
        if( temp->age <= 0 ){
            printf("Error in age input !\n");
            free_stdudents();
            fclose(fp);
            return ERR_AGE;
        }
        if( !(temp->math >=0 && temp->math <=100 &&
          temp->english >=0 && temp->english <=100 &&
          temp->history >=0 && temp->history <=100) )
        {
            printf("Error in grade input !\n");
            free_stdudents();
            fclose(fp);
            return ERR_SCORE;
        }

        if( students ){
            lastp->next = temp;
            temp->num = lastp->num + 1;
            lastp = temp;
        }
        else{
            temp->num = 1;
            students = lastp = temp;
        }

        if( lastp->num == 1 )
          printf("No\tName\tAge\tMath\tEnglish\tHistory\n");
        printf("%d\t%s\t%d\t%d\t%d\t%d\n", lastp->num, lastp->name, lastp->age,
        lastp->math, lastp->english, lastp->history);
    }
    fclose(fp);
    return NORMAL;
}
```

Sample (3/4)

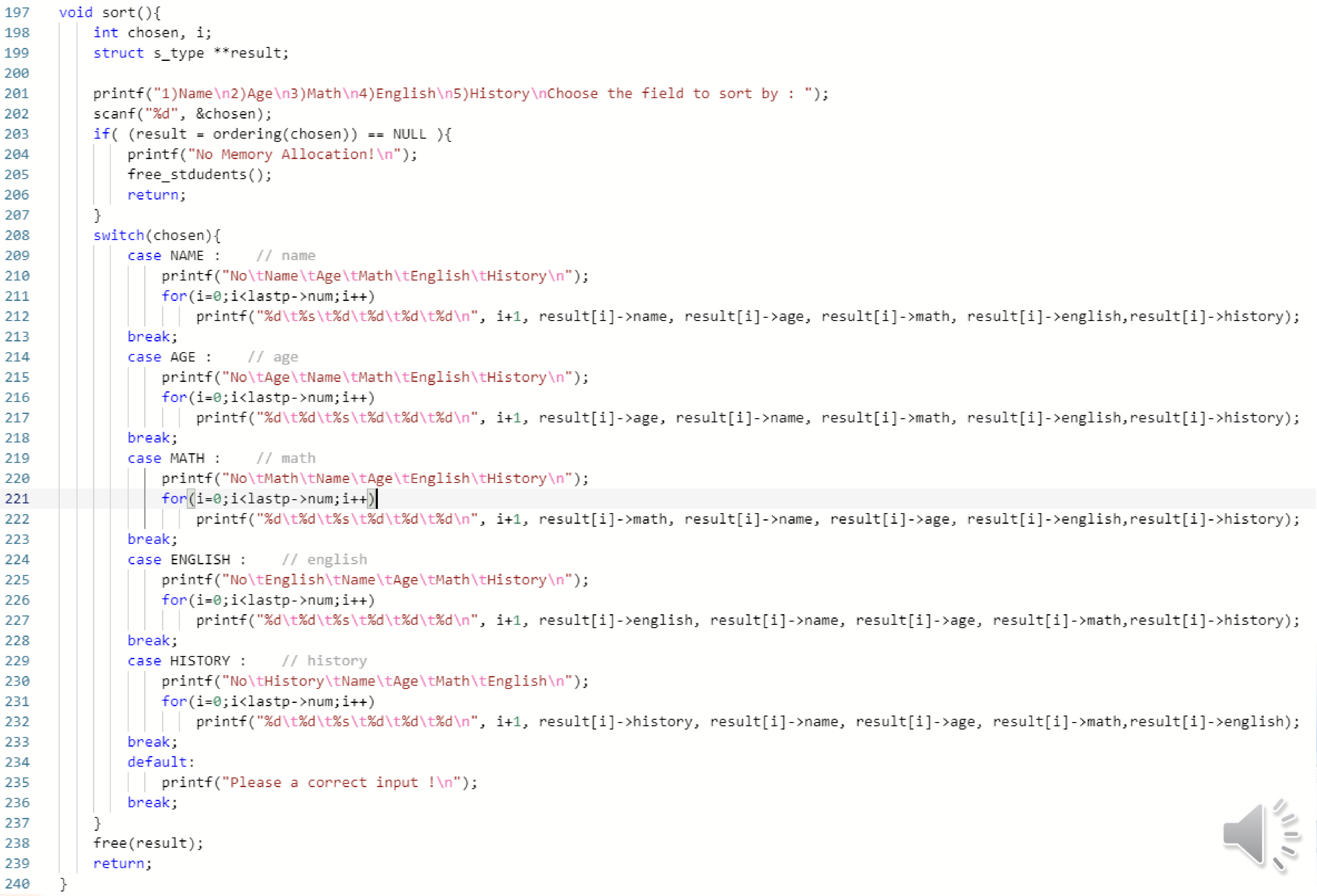


```
struct s_type **ordering( int c){
    struct s_type **nlist, *p, *temp;
    int min, i ;

    if((nlist = (struct s_type **)malloc(sizeof(struct s_type *)*( lastp->num )))!=NULL){
        for(min=0, p=students;p && min<lastp->num ; min++, p=p->next )
            nlist[min] = p;
        for(min=0 ; min<lastp->num ; min++){
            for( i=min+1; i<lastp->num ; i++){
                switch( c ){
                    case NAME :
                        if(strcmp(nlist[min]->name, nlist[i]->name)>0)
                        {
                            temp = nlist[min];
                            nlist[min] = nlist[i];
                            nlist[i] = temp;
                        }
                    break;
                    case AGE :
                        if(nlist[min]->age > nlist[i]->age)
                        {
                            temp = nlist[min];
                            nlist[min] = nlist[i];
                            nlist[i] = temp;
                        }
                    break;
                    case MATH :
                        if(nlist[min]->math > nlist[i]->math)
                        {
                            temp = nlist[min];
                            nlist[min] = nlist[i];
                            nlist[i] = temp;
                        }
                    break;
```



```
                    case ENGLISH :
                        if(nlist[min]->english > nlist[i]->english)
                        {
                            temp = nlist[min];
                            nlist[min] = nlist[i];
                            nlist[i] = temp;
                        }
                    break;
                    case HISTORY :
                        if(nlist[min]->history > nlist[i]->history)
                        {
                            temp = nlist[min];
                            nlist[min] = nlist[i];
                            nlist[i] = temp;
                        }
                    break;
                }
            }
        }
    }

    return nlist;
}
```

Sample (4/4)

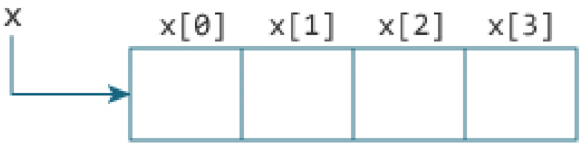


```c
void sort(){
    int chosen, i;
    struct s_type **result;

    printf("1)Name\n2)Age\n3)Math\n4)English\n5)History\nChoose the field to sort by : ");
    scanf("%d", &chosen);
    if( (result = ordering(chosen)) == NULL ){
        printf("No Memory Allocation!\n");
        free_stdudents();
        return;
    }
    switch(chosen){
        case NAME :    // name
            printf("No\tName\tAge\tMath\tEnglish\tHistory\n");
            for(i=0;i<lastp->num;i++)
                printf("%d\t%s\t%d\t%d\t%d\t%d\n", i+1, result[i]->name, result[i]->age, result[i]->math, result[i]->english,result[i]->history);
        break;
        case AGE :    // age
            printf("No\tAge\tName\tMath\tEnglish\tHistory\n");
            for(i=0;i<lastp->num;i++)
                printf("%d\t%d\t%s\t%d\t%d\t%d\n", i+1, result[i]->age, result[i]->name, result[i]->math, result[i]->english,result[i]->history);
        break;
        case MATH :    // math
            printf("No\tMath\tName\tAge\tEnglish\tHistory\n");
            for(i=0;i<lastp->num;i++)
                printf("%d\t%d\t%s\t%d\t%d\t%d\n", i+1, result[i]->math, result[i]->name, result[i]->age, result[i]->english,result[i]->history);
        break;
        case ENGLISH :    // english
            printf("No\tEnglish\tName\tAge\tMath\tHistory\n");
            for(i=0;i<lastp->num;i++)
                printf("%d\t%d\t%s\t%d\t%d\t%d\n", i+1, result[i]->english, result[i]->name, result[i]->age, result[i]->math,result[i]->history);
        break;
        case HISTORY :    // history
            printf("No\tHistory\tName\tAge\tMath\tEnglish\n");
            for(i=0;i<lastp->num;i++)
                printf("%d\t%d\t%s\t%d\t%d\t%d\n", i+1, result[i]->history, result[i]->name, result[i]->age, result[i]->math,result[i]->english);
        break;
        default:
            printf("Please a correct input !\n");
        break;
    }
    free(result);
    return;
}
```

Relationship Between Arrays and Pointers (1/11)

```
#include <stdio.h>
int main() {
   int x[4];
   int i;

   for(i = 0; i < 4; ++i) {
      printf("&x[%d] = %p\n", i, &x[i]);
   }

   printf("Address of array x: %p", x);

   return 0;
}
```

```
&x[0] = 1450734448
&x[1] = 1450734452
&x[2] = 1450734456
&x[3] = 1450734460
Address of array x: 1450734448
```



- &x[0] is equivalent to x.
  - **x[0] is equivalent to *x.**
- &x[1] is equivalent to x+1
  - **x[1] is equivalent to *(x+1).**
- …
- Basically, &x[i] is equivalent to x+i
  - **x[i] is equivalent to *(x+i).**

Relationship Between Arrays and Pointers (2/11)

```
#include <stdio.h>
int main() {
  int i, x[6], sum = 0;
  printf("Enter 6 numbers: ");
  for(i = 0; i < 6; ++i) {
  // Equivalent to scanf("%d", &x[i]);
      scanf("%d", x+i);

  // Equivalent to sum += x[i]
      sum += *(x+i);
  }
  printf("Sum = %d", sum);
  return 0;
}
```

```
Enter 6 numbers:  2
 3
 4
 4
 12
 4
Sum = 29
```

```
#include <stdio.h>
int main() {
  int x[5] = {1, 2, 3, 4, 5};
  int* ptr;

  // ptr is assigned the address of the third element
  ptr = &x[2];  // ptr = x+2;

  printf("*ptr = %d \n", *ptr);   // 3
  printf("*(ptr+1) = %d \n", *(ptr+1)); // 4
  printf("*(ptr-1) = %d", *(ptr-1));  // 2

  return 0;
}
```

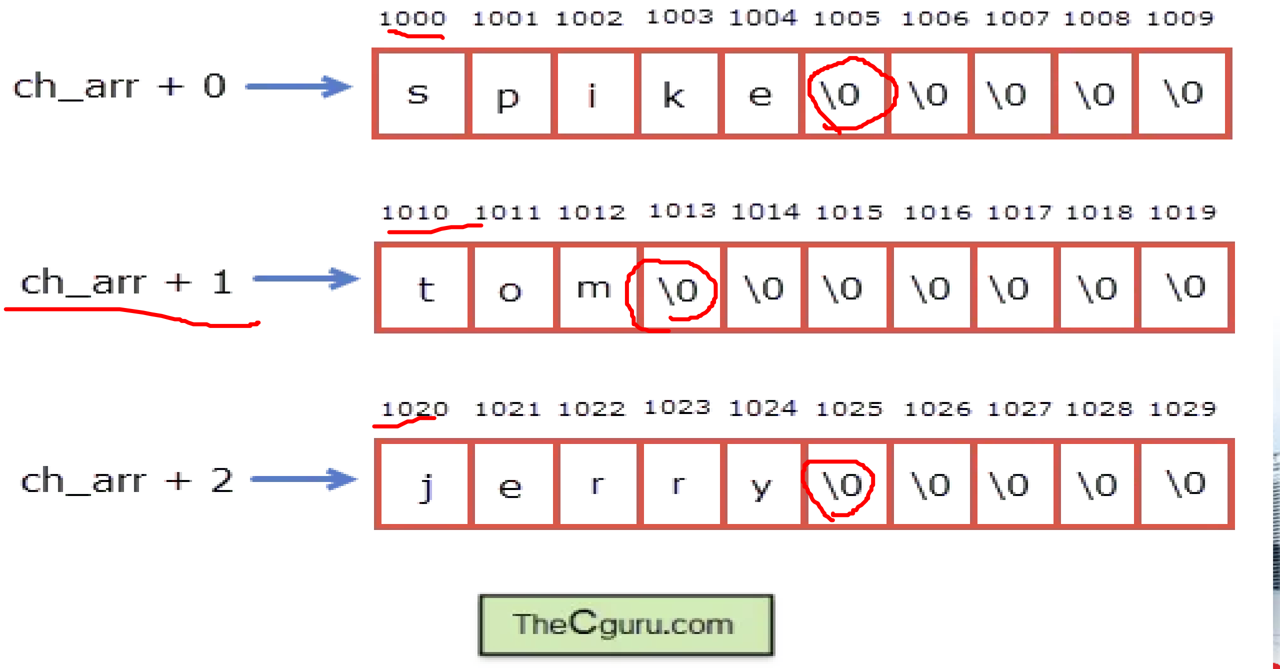
```
*ptr = 3
*(ptr+1) = 4
*(ptr-1) = 2
```

Relationship Between Arrays and Pointers (3/11)

```
char ch_arr[3][10] = {
                        {'s', 'p', 'i', 'k', 'e', '\0'},
                        {'t', 'o', 'm','\0'},
                        {'j', 'e', 'r', 'r', 'y','\0'}
                    };
```

```
char ch_arr[3][10] = {
                        "spike",
                        "tom",
                        "jerry"
                    };
```

- It is important to end each 1-D array by the null character
  - **otherwise, it will be just an array of characters.**
  - **We can't use them as strings.**



	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009
ch_arr + 0 →	s	p	i	k	e	\0	\0	\0	\0	\0

	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019
ch_arr + 1 →	t	o	m	\0	\0	\0	\0	\0	\0	\0

	1020	1021	1022	1023	1024	1025	1026	1027	1028	1029
ch_arr + 2 →	j	e	r	r	y	\0	\0	\0	\0	\0

TheCguru.com

Relationship Between Arrays and Pointers (4/11)

```
#include<stdio.h>

int main()
{
    int i;

    char ch_arr[3][10] = {
                         "spike",
                         "tom",
                         "jerry"
                     };

    printf("1st way \n\n");

    for(i = 0; i < 3; i++)
    {
        printf("string = %s \t address = %u\n", ch_arr + i, ch_arr + i);
    }

    return 0;
}
```

```
string = spike        address = 2686736
string = tom          address = 2686746
string = jerry        address = 2686756
```

Relationship Between Arrays and Pointers (5/11)

```
char ch_arr[3][10] = {
                    {'s', 'p', 'i', 'k', 'e', '\0'},
                    {'t', 'o', 'm','\0'},
                    {'j', 'e', 'r', 'r', 'y','\0'}
                };

ch_arr[0] = "tyke";           // invalid
ch_arr[1] = "dragon";         // invalid

strcpy(ch_arr[0], "type");    // valid ✓
```

ch_arr[0]

[1]

[2]

Relationship Between Arrays and Pointers (6/11)

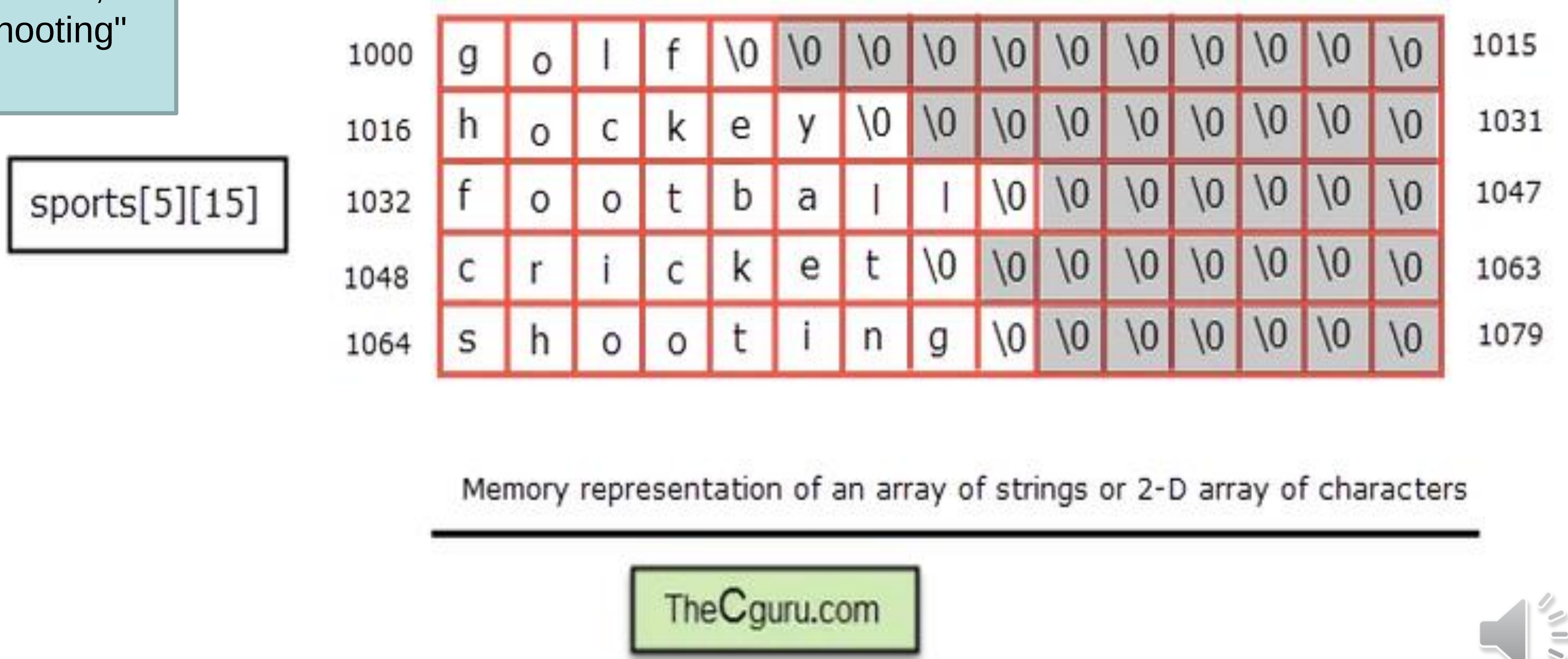
```
#include<stdio.h>
#include<string.h>

int factorial(int n)
{
    if(n == 0)
        return 1;
    else
        return n * factorial(n-1);
}

int main()
{
    int i, found = 0, n;
    char name[10], master_list[5][20] = { "admin", "tom", "bob", "tim", "jim" } ;

    printf("Enter username: ");
    gets(name);
    for(i = 0; i < 5; i++)
        if(strcmp(name, master_list[i]) == 0 )
        {
            found = 1;
            break;
        }

    if(found==1)
    {
        printf("Welcome %s !\n", name);
        printf("Enter a number to calculate the factorial: ");
        scanf("%d", &n);
        printf("Factorial of %d is %d", n, factorial(n));
    }
    else
        printf("Error: You are not allowed to run this program.", name);

    return 0;
}
```

```
Enter username: admin
Welcome admin !
Enter a number to calculate the factorial: 4
Factorial of 4 is 24
```

```
Enter username: jack
Error: You are not allowed to run this program.
```

Relationship Between Arrays and Pointers (7/11)

```
char sports[5][15] = {
                        "golf",
                        "hockey",
                        "football",
                        "cricket",
                        "shooting"
                };
```



sports[5][15]

1000	g	o	l	f	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	1015
1016	h	o	c	k	e	y	\0	\0	\0	\0	\0	\0	\0	\0	\0	1031
1032	f	o	o	t	b	a	l	l	\0	\0	\0	\0	\0	\0	\0	1047
1048	c	r	i	c	k	e	t	\0	\0	\0	\0	\0	\0	\0	\0	1063
1064	s	h	o	o	t	i	n	g	\0	\0	\0	\0	\0	\0	\0	1079

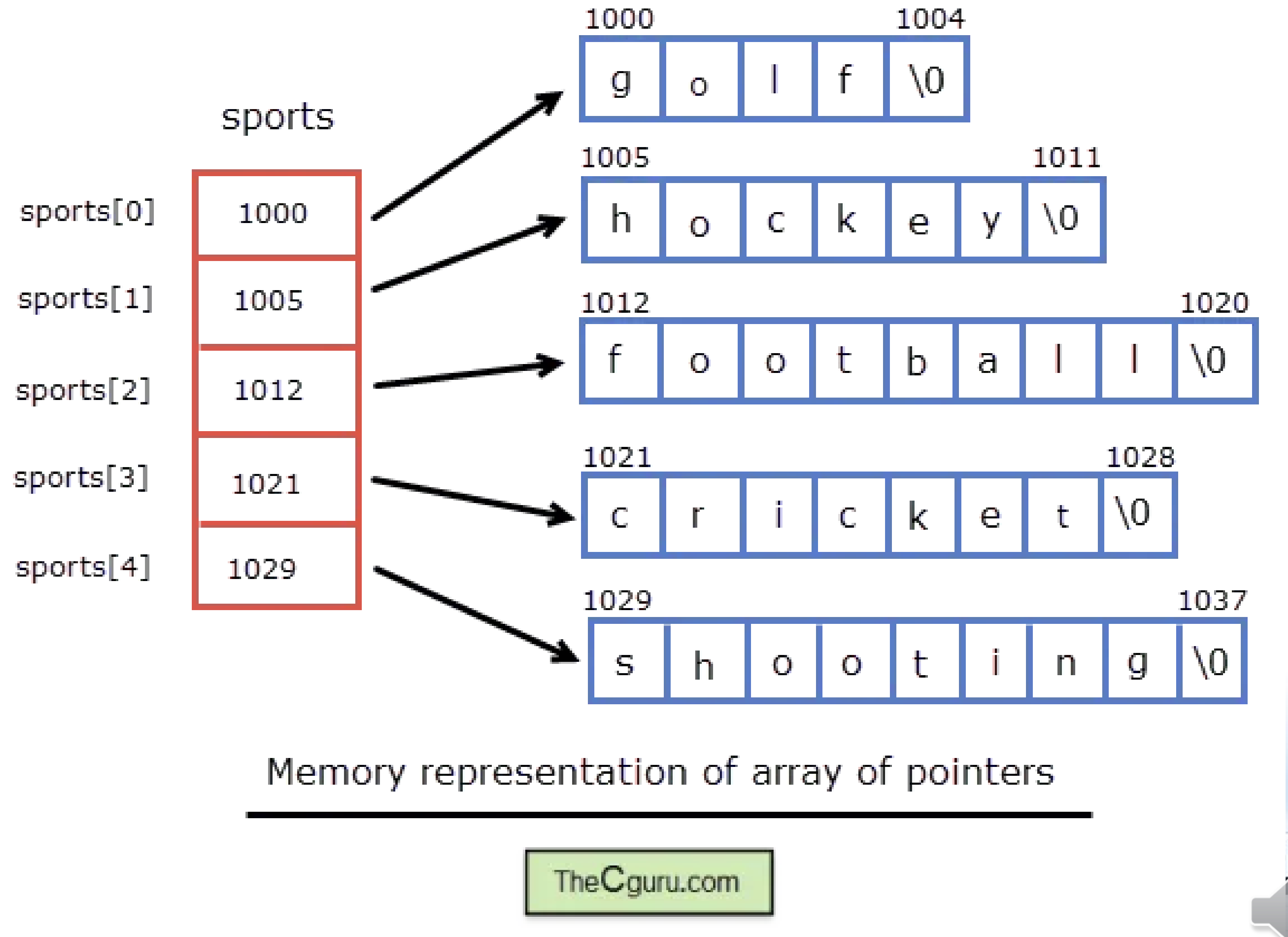
Memory representation of an array of strings or 2-D array of characters

TheCguru.com

Relationship Between Arrays and Pointers (8/11)

```
char *sports[5] = {
                    "golf",
                    "hockey",
                    "football",
                    "cricket",
                    "shooting"
                };
```

```
char *sports[] = {
                    "golf",
                    "hockey",
                    "football",
                    "cricket",
                    "shooting"
                };
```



Memory representation of array of pointers

Relationship Between Arrays and Pointers (9/11)

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

int main()
{
    int i ;

    char *sports[] = { "golf",
                       "hockey",
                       "football",
                       "cricket",
                       "shooting"  };

    for(i = 0; i < 5; i++){
        printf("String = %10s", sports[i] );
        printf("\tAddress of string literal = %u\n", sports[i]);
    }

    return 0;
}
```

```
String = golf        Address of string literal = 4206592
String = hockey      Address of string literal = 4206597
String = football    Address of string literal = 4206604
String = cricket     Address of string literal = 4206613
String = shooting    Address of string literal = 4206621
```

Relationship Between Arrays and Pointers (10/11)

```
char games[3][10] = {
                    "roadrash",
                    "nfs",
                    "angrybirds"
                };

games[0] = "hitman";   // wrong
```

```
char *games[3] = {
                 "roadrash",
                 "nfs",
                 "angrybirds"
             };

games[0] = "hitman";   // ok
```

- can't assign a new string to a 2-D array of characters using assignment operator (=).

Relationship Between Arrays and Pointers (11/11)

```
char *top_games[5];

scanf("%s", top_games[0]);                    ~~// invalid~~
strcpy(top_games[0], "mario");                // invalid
gets(top_games[0]);                           // invalid
strcat(top_games[0], "needforspeed");         // invalid
```