C Programming Language

(7th class)

Dohyung Kim

Assistant Professor @ Department of Computer Science

Today …

- Dynamic Memory Allocation
- Struct
- Union
- Typedef
- #define

Limitations in array

- **The size of the array must be known beforehand**
  - int scores[10][4];
- **The size of the array cannot be changed in the duration of your program**
  - How could you add a new student in your program?

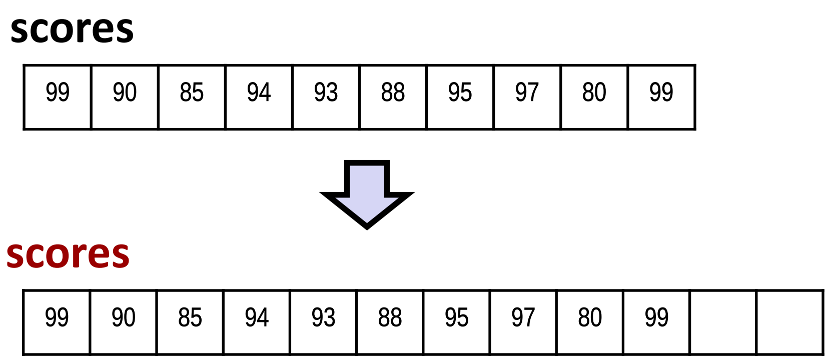
Dynamic Memory Allocation

- **#include<stdlib.h>**
  - void *malloc(int size);
  - void *calloc(int count, int unit_size);
  - void *realloc(void* ptr, int size);
  - void *free(void *ptr);

```
int scores[10];
```

```
int num = 10;
int *scores = malloc(num*sizeof(int));
/* int *scores = calloc(num, sizeof(int)); */
…
/* I want to add two new students */
scores = realloc(scores, (num+2)*sizeof(int));

/* the memory space is released */
free(scores);
```



Struct

- A data structure that combines data items of different kinds

```
struct mystruct{
   char[10] name;
   int id;
   int scores[3];
} student;
```

```
struct mystruct{
   char[10] name;
   int id;
   int scores[3];
} ;
…
struct mystruct student;
```

```
typedef struct {
   char[10] name;
   int id;
   int scores[3];
} Mystruct ;
…
Mystruct student;
```

- How to initialize each piece of data in the struct?

```
Mystruct student = {“Albert”, 20170000, {100, 100, 100}}
```

```
Mystruct student;
strcpy(student.name, “Albert”);
student.id = 20170000;
student.scores[0] = 100;
student.scores[1] = 100;
student.scores[2] = 100;
```

Struct

```
typedef struct mystruct{
   char* name;
   int id;
   int scores[3];
} Mystruct;

Mystruct student;
student.name = “Albert”;
student.id = 20170000;
student.scores[0] = 100;
student.scores[1] = 100;
student.scores[2] = 100;
```

Struct

```
typedef struct mystruct{
   char* name;
   int id;
   int scores[3];
} Mystruct;

void main() {
   Mystruct student;
   student.name = “Albert”;
   student.id = 20170000;
   student.scores[0] = 100;
   student.scores[1] = 100;
   student.scores[2] = 100;
   …
}
```

```
struct score{
   int math;
   int physics;
   int English;
   double average;
};

typedef struct mystruct{
   char* name;
   int id;
   struct score myScore;
} Mystruct;

void main() {
   Mystruct student;
   student.name = “Albert”;
   student.id = 20170000;
   student.myScore.math = 100;
   student.myScore.physics = 100;
   student.myScore.English = 100;
   ….
}
```

Malloc and Struct

```
struct score{
   int math;
   int physics;
   double average;
};

typedef struct mystruct{
   char* name;
   int id;
   struct score myScore;
} Mystruct;

void main() {
   Mystruct* student;
   student = malloc(sizeof(Mystruct));
   student->name = “Albert”;
   student->id = 20170000;
   student->myScore.math = 100;
   student->myScore.physics = 100;
   ….
 }
```

```
(*students).name = “Albert”;

students->name = “Albert”
```

Malloc and Struct

```
struct score{
   int math;
   int physics;
};

typedef struct mystruct{
   char name[10];
   int id;
   struct score myScore;
} Mystruct;
```

```
void main() {
   int i=0;
   Mystruct* student;
   student = malloc(sizeof(Mystruct));
   char more[2];

   do {
      ++i;
      student = realloc(student, i*sizeof(Mystruct));
      printf("input name : ");
      scanf("%s", student[i-1].name);
      printf("input id : ");
      scanf("%d", &student[i-1].id);
      printf("input math score : ");
      scanf("%d", &student[i-1]. myScore.math);
      printf("input math score : ");
      scanf("%d", &student[i-1].myScore.math);
      printf("Do you want to store more data? (y/n)");
      scanf("%s", more);
   }while(!strcmp(more, "y"));
   …
   for (i = 0; i < num; i++) {
      free(student);
   }
}
```

Malloc and Struct

```
struct score{
   int math;
   int physics;
};

typedef struct mystruct{
   char name[10];
   int id;
   struct score myScore;
} Mystruct;
```

```
void main() {
   int i=0;
   Mystruct *student, *tmp = NULL;
   student = malloc(sizeof(Mystruct));
   char more[2];

   do {
      ++i;
      student = realloc(student, i*sizeof(Mystruct));
      if(tmp == NULL)  tmp = student;
      else  tmp++;

     printf(“input name : ”);
      scanf(“%s”, tmp->name);
      printf(“input id : ”);
      scanf(“%d”, &tmp->id);
      printf(“input math score : ”);
      scanf(“%d”, &tmp->myScore.math);
      printf(“input math score : ”);
      scanf(“%d”, &tmp->myScore.math);
      printf(“Do you want to store more data? (y/n)”);
      scanf(“%s”, more);
   }while(!strcmp(more, “y”));
   ….
```

Union

- A data structure that allows to store different data types in the same memory location

```
union myunion{
   char[10] name;
   int id;
   int scores[3];
} student;
```

```
union myunion{
   char[10] name;
   int id;
   int scores[3];
} ;
…
union myunion student;
```

```
typedef union{
   char[10] name;
   int id;
   int scores[3];
} Myunion ;
…
Myunion student;
```

- What is the difference between struct and union?

Struct vs. Union

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

struct mystruct{
   int i;
   float f;
   char str[20];
};

int main () {
   struct mystruct data;
   data.i = 10;
   data.f = 220.5;
   strcpy(data.str, “C programming”);

   printf( “data.i : %d\n”, data.i);
   printf( “data.f : %f\n”, data.f);
   printf( “data.str : %s\n”, data.str);

   return 0;
}
```

Output

Struct vs. Union

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

struct mystruct{
   int i;
   float f;
   char str[20];
};

int main () {
   struct mystruct data;
   data.i = 10;
   data.f = 220.5;
   strcpy(data.str, “C programming”);

   printf( “data.i : %d\n”, data.i);
   printf( “data.f : %f\n”, data.f);
   printf( “data.str : %s\n”, data.str);

   return 0;
}
```

Output

```
data.i : 10
data.f : 220.500000
data.str : C Programming
```

printf (“size of struct : %d \n” , sizeof(data));

➔

Struct vs. Union

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

struct mystruct{
   int i;
   float f;
   char str[20];
};

int main () {
   struct mystruct data;
   data.i = 10;
   data.f = 220.5;
   strcpy(data.str, “C programming”);

   printf( “data.i : %d\n”, data.i);
   printf( “data.f : %f\n”, data.f);
   printf( “data.str : %s\n”, data.str);

   return 0;
}
```

Output

```
data.i : 10
data.f : 220.500000
data.str : C Programming
```

printf (“size of struct : %d \n” , sizeof(data));

➔ 28

Struct vs. Union

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

union myunion{
   int i;
   float f;
   char str[20];
};

int main () {
   union myunion data;
   data.i = 10;
   data.f = 220.5;
   strcpy(data.str, “C programming”);

   printf( “data.i : %d\n”, data.i);
   printf( “data.f : %f\n”, data.f);
   printf( “data.str : %s\n”, data.str);

   return 0;
}
```

Output

Struct vs. Union

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

union myunion{
   int i;
   float f;
   char str[20];
};

int main () {
   union myunion data;
   data.i = 10;
   data.f = 220.5;
   strcpy(data.str, “C programming”);

   printf( “data.i : %d\n”, data.i);
   printf( “data.f : %f\n”, data.f);
   printf( “data.str : %s\n”, data.str);

   return 0;
}
```

Output

What?

```
data.i : 1917853763
data.f : 4122360580327794860452759994368.000000
data.str : C Programming
```

```
printf (“size of union : %d \n” , sizeof(data));
```

➔

Struct vs. Union

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

union myunion{
   int i;
   float f;
   char str[20];
};

int main () {
   union myunion data;
   data.i = 10;
   data.f = 220.5;
   strcpy(data.str, “C programming”);

   printf( “data.i : %d\n”, data.i);
   printf( “data.f : %f\n”, data.f);
   printf( “data.str : %s\n”, data.str);

   return 0;
}
```

Output

What?

```
data.i : 1917853763
data.f : 4122360580327794860452759994368.000000
data.str : C Programming
```

```
printf (“size of union : %d \n” , sizeof(data));
```

➔ 20

Aha!

Use case of UNION

```
typedef union {
   struct {
      unsigned char byte1;
      unsigned char byte2;
      unsigned char byte3;
      unsigned char byte4;
   }bytes;
   unsigned int dword;
}HW_Register;

….

HW_Register reg;

reg.dword = 0x12345678;
Printf(“value in byte3: %d\n”, reg.bytes.byte3);
/*byte3 : 0x34 – printf shows the value of 16*3+4 =
86*/
```

Access the register directly

```
#define KEY_RIGHT    (1<<0)  /*00000001*/
#define KEY_LEFT     (1<<1)  /*00000010*/
#define KEY_DOWN  (1<<2)  /*00000100*/
#define KEY_UP       (1<<3)  /*00001000*/

typedef union{
   struct {
      unsigned char right:1;
      unsigned char left:1;
      unsigned char down:1;
      unsigned char up:1;
      unsigned char reserved:4;
   }keyBits;
   unsigned char byte;
}KEYS;

void main() {
   unsigned char value;
   KEYS input;
   while(1){
      printf(“input your number: ”);
      scanf(“%d”, &value);
      input.byte = value;
      if(input.keyBits.right) printf(“user pressed key right\n”);
      if(input.keyBits.left) printf(“user pressed key left\n”);
      if(input.keyBits.down) printf(“user pressed key down\n”);
      if(input.keyBits.up) printf(“user pressed key up\n”);
   }
   return;
}
```

Access a single bit

Typedef

- typedef is used to give a type, a new name.

```
typedef unsigned int MYINT;

MYINT val1, val2;
/* unsigned int val1, val2; */
```

- You already used typedef to give a name to your user defined data type

```
typedef struct {
   char[10] name;
   int id;
   int scores[3];
} Mystruct ;
…
Mystruct student;
```

#define

- **#define is also used to define the aliases for various data type similar to typedef.**
- **Differences between typedef and #define**
  - typedef is limited to giving symbolic names to types only whereas #define can be used to define alias for values as well.
  - typedef interpretation is performed by the compiler whereas #define statements are processed by the pre-processor.

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0

int main() {
   printf("Value of TRUE: %d\n", TRUE);
   printf("Value of FALSE: %d\n", FALSE);
   return 0;
}
```

```
typedef char* STR;

#define STR char*
```

Q and A