Advanced C Programming & Lab

11. Structures

Sejong University

Outline

1) **Structures?**
2) Declaration and Operations
3) Arrays of Structures
4) Structures and Pointers
5) Structures and Functions
6) typedef

1) Structures?

- **Suppose we need to process students' information**
  - Students' information: ID, Name, GPA
  - Need three variables

```
int id;
char name[8];
double grade;
```

  - Need to process 100 students' information?
    - ✓ Method 1: **Array**

```
int id[100];
char name[100][8];
double grade[100];
```

1) Structures?

- Need to process 100 students' information?
  - ✓ Method 2: **Group each student's information**
    - ✓ Arrays cannot handle them together due to the differences in data type
    - ✓ **Structures** can do this!

```
struct student{
    int id;
    char name[8];
    double grade;
} st[100];
```

1) Structures?

- **Structures**
  - Data type that groups multiple variables of differing data types together
    - ✓ Similar to other data types such as int, char
    - ✓ int, char are pre-defined, but structures are use-defined data types for specific purposes
  - Variables in a structure are called **members**

1) Structures?

- **Defining a Structure**
  - Define a data type
  - Use a keyword: struct
  - Examples

```
struct Name{
    member1 declaration;
    member2 declaration;
    ...
};
```

```
struct student{
    int id;
    char name[8];
    double grade;
};
```

  - Do not allocate a space in a memory yet

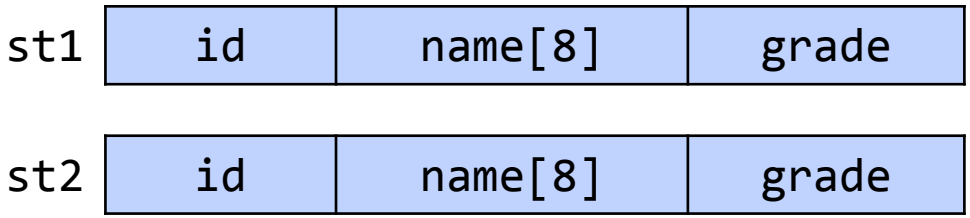
1) Structures?

- **Declaration of structure variables**
  - Similar to normal variables

    Data type Variable name;

```
struct student{     // Structure Declaration
   int id;
   char name[8];
   double grade;
};

struct student st1, st2;  // Structure Variable Declaration
```

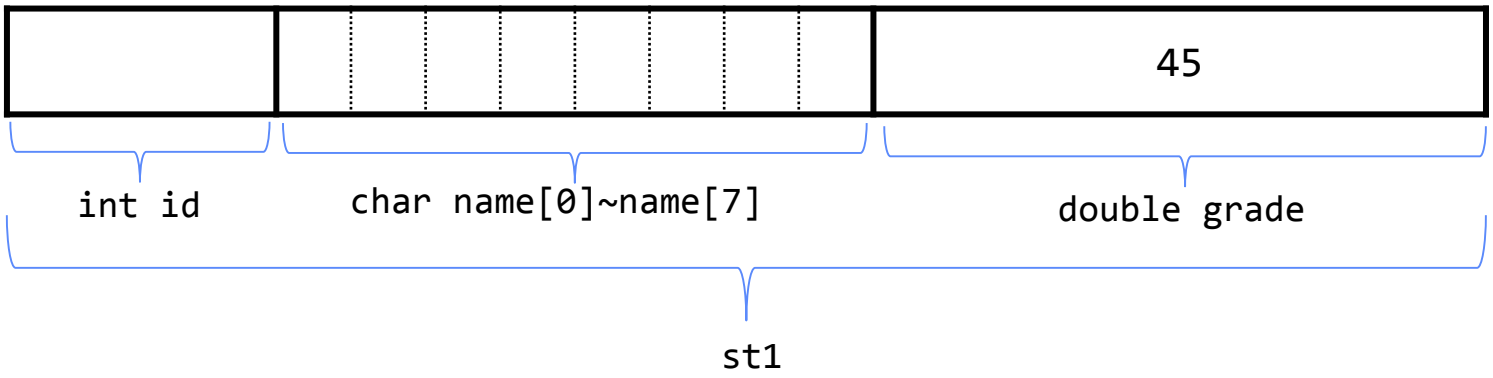
  - Allocate a space in a memory now!



1) Structures?

- **Memory allocation**

```
struct student{      // Structure Declaration
   int id;    char name[8];    double grade;
};

struct student st1;  // Structure Variable Declaration
```



A structure variable itself does not take a space, but its members do have in a memory.

1) Structures?

- **Access to structure members**
  - Access operator (.) : Structure variable . **Member variables**
  - Structure members can have differing data types
    - ✓ int type member is used just like a normal int variable

```
struct student st1, st2;   // Structure
                           // variable declaration

st1.id = 10;
st1.id = st1.id * 2;
printf("id: %d", st1.id);
```

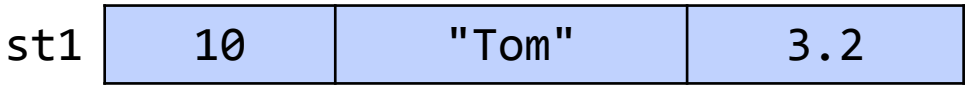
1) Structures?

- **Structure member initialization**
  - Use { }
  - According to the order of member declaration

```
struct student st1 = { 10, "Tom", 3.2};

printf("id: %d\n", st1.id);
printf("name: %s\n", st1.name);
printf("grade: %.2f\n", st1.grade);
```

Results

```
id: 10
name: Tom
grade: 3.20
```



1) Structures?

- **Example 1:**

```
struct student{      // Define student structure
   int id;           // outside function defition
   char name[8];
   double grade;
};

void main( )
{  struct student st1 = {10, "Tom", 3.2};
   // Declaration and Initialization

   st1.id += 20;                  // structure member access
   strcpy(st1.name, "alice");     // Caution: st1.name = "alice" (X)
   st1.name[0] = 'A';

   printf("id: %d\n", st1.id);        // Access to structure members
   printf("name: %s\n", st1.name);
   printf("grade: %.2f\n", st1.grade);
}
```

Results

```
id: 30
name: Alice
grade: 3.20
```

1) Structures?

- **Summary of Structures**

	variable	array	**structure**
Declaration	`int a;`	`int a[3];`	**`struct student st;`**
Access	`a = 10`	`a[0] = 10;`	**`st.id = 10;`**
Initialization	`Int a=10;`	`int a[3]={1,2,3};`	**`struct student st={10,"Tom",3.2};`**

1) Structures?

- **[Practice 1] Receive prices of 1 maindish, 3 sidedish, 1 drink. Print the total price.**

  1. Use normal variables and an array: maindish, sidedish[3], beverage

  2. Use a structure lunchbox containing maindish, sidedish, drink

Example

```
Main dish: 30
Side dish 1: 3
Side dish 2: 5
Side dish 3: 0
Beverage: 10

Total: 30 + 3 + 5 + 0 + 10 = 48
```

1) Structures?

- **How many bytes a structure takes?**
  - Use sizeof()
    - ✓ 4 (int) + 8 (char array) + 8 (double) = 20 bytes ?

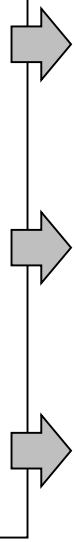
```
struct student{
   int id;   char name[8];   double grade;
};

void main( )
{  printf("%d ", sizeof( int ) );
   printf("%d ", sizeof( char[8]) );
   printf("%d ", sizeof( double ) );
   printf("%d ", sizeof( struct student ) );
}
Results:
4 8 8 24
```

1) Structures?

- Change structure members

Size (Byte)

```
struct student1{
   int id;                    double grade;
};

struct student2{
   int id;   char name[3];    double grade;
};

struct student3{
             char name[3];    double grade;
};
```



16

16

16

➔ **Memory allocation: a multiple of 8**

- ✓ May depend on a computer system
- ✓ Remaining space is unused

Outline

1) Structures?
2) **Declaration and Operations**
3) Arrays of Structures
4) Structures and Pointers
5) Structures and Functions
6) typedef

2) Declaration and Operations

- **Definition and Declaration 1**
  - Separate definition and declaration

```
struct student{     // Definition
   int id;   char name[8];   double grade;
};

void func1(){
   struct student st1;  // Declaration
   ...
}
void func2(){
   struct student st2;  // Declaration
   ...
}
```

2) Declaration and Operations

- **Definition and Declaration 2**
  - Simultaneous definition and declaration
    - ✓ **student** : structure name, **st** : variable name

```
struct student{      // definition
   int id;   char name[8];   double grade;
} st;              // declaration: a variable st (global)

void func1(){
   struct student st1;  // declaration (local)
   ...
}
void func2(){
   struct student st2;  // declaration (local)
   ...
}
```

2) Declaration and Operations

- **Definition and Declaration 3**
  - Simultaneous definition, declaration, and initialization

```
struct student{      // definition
   int id;   char name[8];   double grade;
} st = {10, "Tom", 3.2} ;  // declaration and initialization

void func1(){
   struct student st1;  // declaration (local)
   ...
}
void func2(){
   struct student st2;  // declaration (local)
   ...
}
```

2) Declaration and Operations

- **Omitting structure name?**
  - No problem with declaration, but cannot be reused

```
struct student{      // definition
   int id;   char name[8];   double grade;
} st;              // declaration: st (o.k.)

void func1(){
   struct student st1, st2;  // compilation error (X)
   ...
}
void func2(){
   struct student st1, st2;  // compilation error (X)
   ...
}
```

2) Declaration and Operations

- **Define in a function?**
  - ✓ Can be used within the function

```
void func1(){
   struct student{     // definition (inside a function)
      int id;   char name[8];   double grade;
   };
   struct student st1;  // declaration
   ...
}
void func2(){
   struct student st2;  // compilation error (X)
   ...
}
```

2) Declaration and Operations

- **Operations**
  - Structures are user-defined data types
    Operations are limited
    - ✓ Ex) mathematical, comparison operations are not supported
      - ✓ st1 + st2 : unclear
      - ✓ st1 < st2 : unclear
  - Operations that are available
    - ✓ assignment, &, sizeof …

2) Declaration and Operations

- **Assignment**
  - Structure members

```
struct student st1 = { 10, "Tom", 3.2};
struct student st2;

st2 = st1;
printf("id: %d\n", st2.id);
printf("name: %s\n", st2.name);
printf("grade: %.2f\n", st2.grade);
```

Results

```
id: 10
name: Tom
grade: 3.20
```

2) Declaration and Operations

- **Assignment**
  - Two are equivalent

```
st2 = st1;
```

=

```
st2.id = st1.id;
st2.name[0] = st1.name[0];
st2.name[1] = st1.name[1];
...
st2.name[8] = st1.name[8];
st2.grade = st1. grade;
```

2) Declaration and Operations

- **[Practice 2] Use the lunchbox structure in Practice 1**
  - Declare two lunchboxes A and B
  - Receive A's members from a user
  - Copy A to B
  - Receive B's maindish price from a user
  - Print the prices of A and B

Outline

1) Structures?
2) Declaration and Operations
3) **Arrays of Structures**
4) Structures and Pointers
5) Structures and Functions
6) typedef

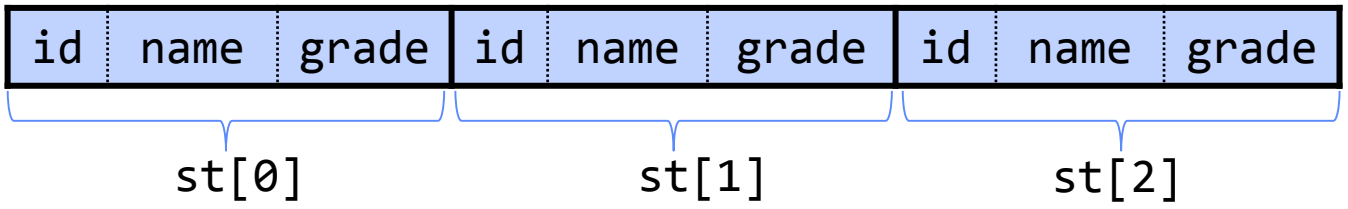
3) Arrays of Structures

- **Arrays of Structures**
  - Similar to normal arrays
    - ✓ Can group the same structures

3) Arrays of Structures

- **Arrays of Structures Declaration**
  - Use [ ]

```
struct student{     // definition
   int id;
   char name[8];
   double grade;
};

struct student st[3]; // declaration
```



3) Arrays of Structures

- **Access to Arrays of Structures**
  - Use [ ]

```
struct student st[3];

st[0].id = 10;
strcpy(st[0].name , "Tom");
st[0].grade = 3.2;

st[1] = st[0];                  // Assignment
st[1].name[2] = 'p';

printf("%d,%s,%.2f\n", st[0].id, st[0].name, st[0].grade);
printf("%d,%s,%.2f\n", st[1].id, st[1].name, st[1].grade);
```

Results

```
10,Tom,3.20
10,Top,3.20
```

3) Arrays of Structures

- **Arrays of Structures Initialization**
  - Use { }

```
int i;
struct student st[3]
    = { { 10, "Tom", 3.2},
      { 20, "Alice"} };
  // remaining members are initialized to 0

for( i= 0 ; i < 3 ; ++i )
  printf("%d:%d,%s,%.2f\n",
        i, st[i].id, st[i].name, st[i].grade);
```

Results

```
0:10,Tom,3.20
1:20,Alice,3.20
2:0,,0.00
```

3) Arrays of Structures

- **Example 2: Complex number addition**

```
struct complex {          // definition
    double real, imag;
};

void main( )
{  int i;
   struct complex x[3] = { {1.2, 2.0} , {-2.2, -0.3} };

   x[2].real = x[0].real + x[1].real;
   x[2].imag = x[0].imag + x[1].imag;

   for( i=0; i < 3 ; ++i )
      printf("x[%d]: %.1f + %.1fi\n", i, x[i].real, x[i].imag);
}
```

3) Arrays of Structures

- **[Practice 3] Use a lunchbox structure in Practice 1. Declare arrays of structures. Receive information of two lunchboxes from a user and print them.**

Outline

1) Structures?
2) Declaration and Operations
3) Arrays of Structures
4) **Structures and Pointers**
5) Structures and Functions
6) typedef

4) Structures and Pointers

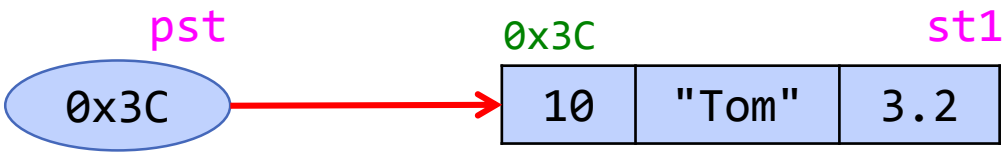
- **Structures and Pointers**
  - Each structure have an address
  - int pointer
    - ✓ Stores an address of int variable
    - ✓ points to an int variable
  - structure pointer
    - ✓ Stores an address of a structure variable
    - ✓ points to a structure variable

  - Same with normal pointer variables
    - ✓ Some expressions are only available for structure pointers

4) Structures and Pointers

- **Structures and Pointers Declaration**
  - Use *
  - &: starting address of a structure variable

```
struct student st1 = { 10, "Tom", 3.2};
struct student *pst; // structure pointer declaration
pst = &st1;    // assignment
```



4) Structures and Pointers

- ***: structure variables**

```
struct student st1 = { 10, "Tom", 3.2}, st2;
struct student *pst = &st1; // declaration and assignment

st2 = *pst; // assignment: structure pointed by pst to st2

printf("%d,%s,%.2f\n", st1.id, st1.name, st1.grade);
printf("%d,%s,%.2f\n", st2.id, st2.name, st2.grade);
```

Results

```
10,Tom,3.20
10,Tom,3.20
```

4) Structures and Pointers

- *** : access to structure members (method 1)**
  - *: access to structure variables
  - .: access to structure members

```
struct student st1, *pst = &st1;

(*pst).id = 20;  // assign 20 to id of
                 // the structure pointed by pst
printf("id: %d\n", (*pst).id + 15 );
```

Results

```
id: 35
```

  - Precedence: . > * (use parenthesis)

4) Structures and Pointers

- *** : access to structure members (method 2)**
  - **->** : only in structure pointers

```
(*pst).id = 20;
```

=

```
pst->id = 20;
```

4) Structures and Pointers

- **Example 3 : Example1(p.11)**

```
struct student{     // student definition
   int id;   char name[8];   double grade;
};

void main( )
{  struct student st1 = {10, "Tom", 3.2};  // declaration, initialization
   struct student *pst = &st1;            // declaration, assignment

   pst->id += 20;         // Access to members
   strcpy(pst->name, "Alice");  // Caution: pst->name = "Alice" (X)
   pst->name[0] = 'A';

   printf("id: %d\n", pst->id);
   printf("name: %s\n", pst->name);
   printf("grade: %.2f\n", pst->grade);
}
```

Results

```
id: 30
name: Alice
grade: 3.20
```

4) Structures and Pointers

- **[Practice 4] Use structure pointers**
  - Define a structure lunchbox including 1 maindish, 3 sidedish, 1 drink
  - Declare structure variables and pointers
  - Receive information of one lunchbox from a user
  - Print them

    ✓ Refer: Example 3(p.39)

4) Structures and Pointers

- **Comparison**

	int pointer	Structure pointer	structure

Declaration	`int *pi, i;`	`struct student{`
		`  ...`
		`} *pst, st;`
			`struct student {`
			`  ...`
			`} st;`

Assignment	`pi=&i;`	`pst=&st;`	
Usage	`*pi = 10;`	`pst->id=10;`	`st.id=10;`
	`printf("%d",*pi);`	`printf("%d",pst->id);`	`printf("%d",st.id);`

Outline

1) Structures?
2) Declaration and Operations
3) Arrays of Structures
4) Structures and Pointers
5) **Structures and Functions**
6) typedef

5) Structures and Functions

- **Structure variables as function parameters**
  - Actual arguments are passed to formal arguments

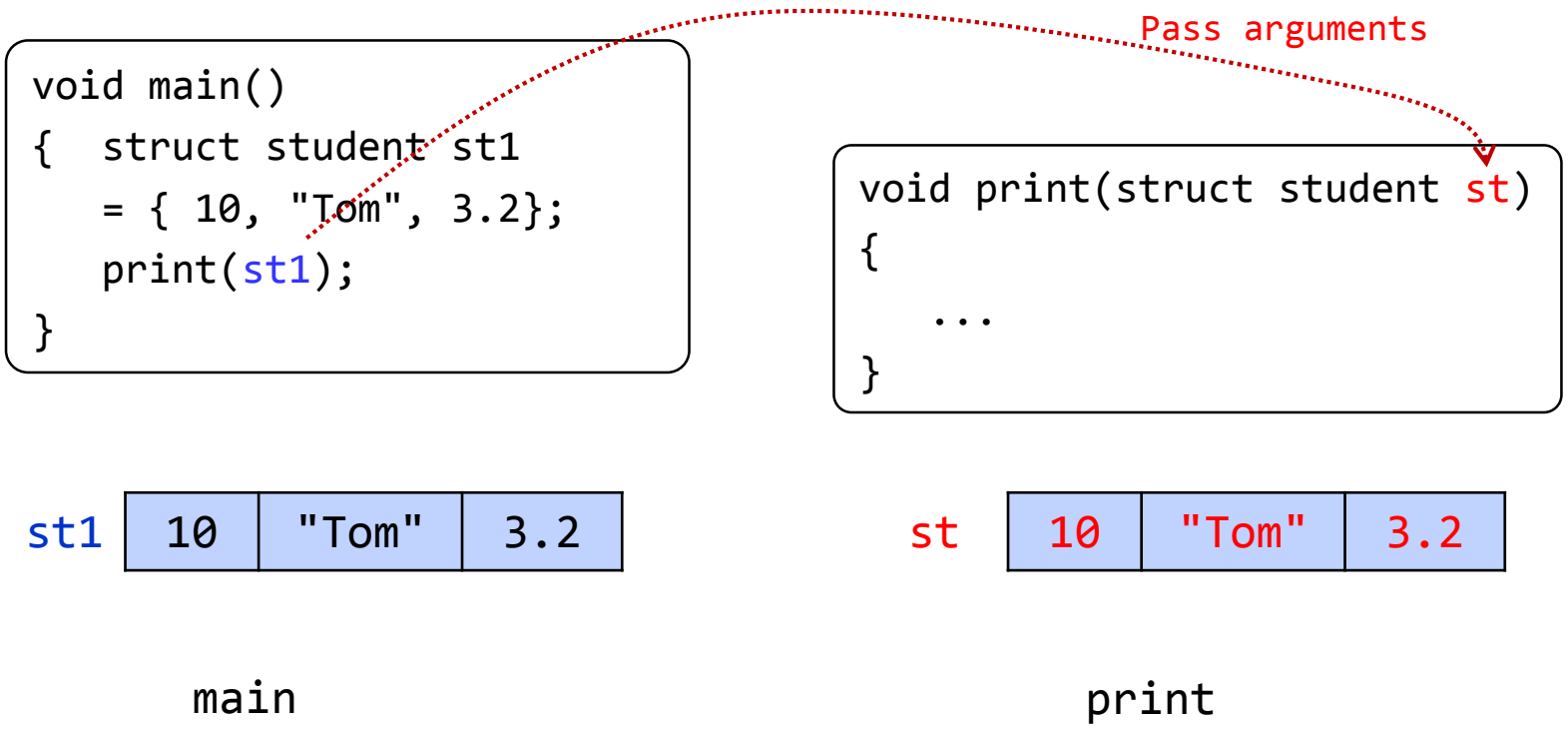
```
void print(struct student st)
{  printf("id: %d\n", st.id);
   printf("name: %s\n", st.name);
   printf("grade: %.2f\n", st.grade);
}

void main()
{  struct student st1 = {10, "Tom", 3.2};
   print(st1);
}
```

Results

```
id: 10
name: Tom
grade: 3.20
```

5) Structures and Functions

- Pass arguments



5) Structures and Functions

- **Use structures as return type**
  - Pass the whole structure

```
struct student init( )
{  struct student st = { 0, "", 0};
   return st;
}
void main()
{  struct student st1 = {10, "Tom", 3.2};
   printf("%d,%s,%.2f\n", st1.id,
               st1.name, st1.grade);
   st1 = init();
   printf("%d,%s,%.2f\n", st1.id,
            st1.name, st1.grade);
}
```

Results

```
10,name, 3.20
0,,0.00
```

5) Structures and Functions

- **How the program below works?**

```
void print(struct student st)
{ ...   // omitted     }

struct student init( )
{ ...   // omitted     }

void main()
{  struct student st1 = {10, "Tom", 3.2};
   print(st1);
   st1 = init();
   print(st1);
}
```

5) Structures and Functions

- **[Practice 5] Receive two complex numbers. Calculate and print their sum.**
  - Use a structure complex (refer: example 2, p.31)
  - Define a function add_complex
    - ✓ Receive two complex structures as arguments, Return their sum as a complex structure
  - main
    - ✓ Receive two complex numbers from a user and store them as structure variables
    - ✓ Call add_complex
    - ✓ Print the sum

5) Structures and Functions

- **Use structure pointer variables as function parameters**
  - Pass actual arguments (address) to formal arguments

```
void init_p(struct student *pst)
{  pst->id = 0;
   pst->name[0] = '\0';
   pst->grade = 0.0;
}
void main()
{  struct student st1 = {10, "Tom", 3.2};
   init_p(&st1);
   printf("%d,%s,%.2f\n", st1.id,
           st1.name, st1.grade);
}
```

Results

```
0,,0.00
```

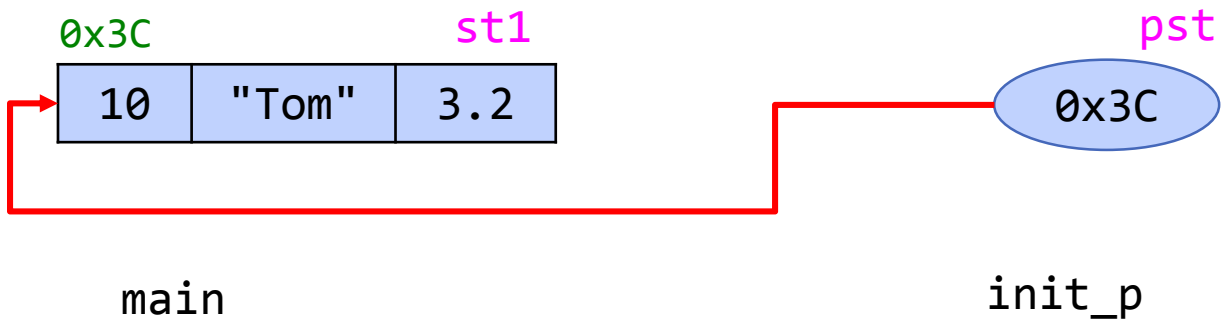
5) Structures and Functions

- Pass arguments

```
void main()
{   struct student st1
    = { 10, "Tom", 3.2};
    init_p(&st1);
}
```

Pass arguments

```
void init_p(struct student *pst)
{
    ...
}
```



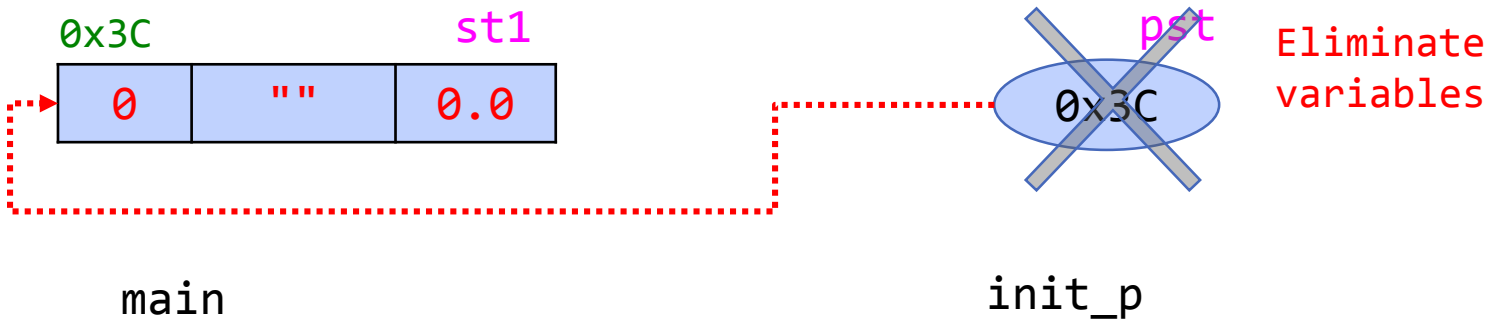
Memory diagram:
calling init_p

5) Structures and Functions

- Terminating a function, local variables are removed

```
void main()
{  struct student st1
   = { 10, "Tom", 3.2};
   init_p(&st1);
}
```

```
void init_p(struct student *pst)
{  pst->id = 0;
   pst->name[0] = '\0';
   pst->grade = 0.0;
}
```



Memory diagram:
terminating init_p

5) Structures and Functions

- **Return the address of a structure variable**

```
struct student *next_addr(struct student *pst)
{
  return pst+1;
}
void main()
{  struct student st[2] = {{10, "Tom", 3.2},
                           {20, "Ann", 3.5}};
   struct student *p;
   p = next_addr(st);
   printf("%d,%s,%.2f\n", p->id,
           p->name, p->grade);
}
```

Results

```
20,Ann,3.50
```

5) Structures and Functions

- **[Practice 6] Receive two complex numbers. Print the complex number whose absolute value is bigger than the other**
  - Use a structure complex (refer: example 2, p.31)
  - Define a function larger_complex
    - ✓ Receive two complex structure pointers, return the structure pointer whose absolute value is bigger than the other
    - ✓ Ex) absolute value of $a+bi$ is a^2+b^2
  - main
    - ✓ Receive complex number from a user and store them as structure variables
    - ✓ Call larger_complex
    - ✓ Print the complex number whose absolute value is bigger than the other

5) Structures and Functions

- **Function call**
  - Function call procedure is identical regardless of parameters, return type, …
  - Pass actual arguments to **formal arguments**
    - ✓ It matters whether it is an integer, character, address?

Outline

1) Structures?
2) Declaration and Operations
3) Arrays of Structures
4) Structures and Pointers
5) Structures and Functions
6) **typedef**

6) typedef

- **typedef**
  - Define a new data type
  - Ex

```
typedef int INT;    // INT data type
```

    - ✓ Give an alternate name INT for int data type
    - ✓ Caution!! INT is not a name of a variable but a name of a data type
    - ✓ Variable declaration using INT

```
INT num;          // INT type variable num
INT arr[5];       // INT type array arr
INT *pi;          // INT type pointer variable pi
```

6) typedef

- **Example**

```
typedef int INT;
typedef int * INTPTR;
typedef unsigned int AGE;
typedef unsigned int ID;
typedef unsigned char UCHAR;
typedef unsigned char * UCHARPTR;

AGE age1, age2;
ID id1, id2;
```

✓ AGE, ID are unsigned int data type

✓ Shorten long names

6) typedef

- **Structures and typedef**
  - Structure: should use a keyword struct
  - Use typedef to shorten data type declarations

```
struct student{
   int id;   char name[8];   double grade;
};
typedef struct student STUDENT;  // definition
STUDENT st;      // STUDENT data type: variable st
```

✓ Define a new data type STUDENT for struct student

✓ Caution!! struct student and STUDENT are data types

6) typedef

- **Example**

```
struct student{
   int id;  char name[8];  double grade;
};

typdef struct student STUDENT;   // definition

void main( )
{  STUDENT st1 = {10, "Tom", 3.2};

   printf("id: %d\n", st1.id);       //access to members
   printf("name: %s\n", st1.name);
   printf("grade: %.2f\n", st1.grade);
}
```

6) typedef

- **definition**
  - Structure definition and typedef together

```
typedef struct student{
   int id;   char name[8];   double grade;
} STUDENT;        // definition
STUDENT st;      // declaration: STUDENT type variable st
```

```
typedef struct student{
   int id;   char name[8];   double grade;
} student;  // structure and user-defiend data type
            // can have the same name
student st;      // declaration: student type variable st
```

```
typedef struct student{     // Omit structure name
   int id;   char name[8];   double grade;
} student;
student st;      // declaration: student type variable st
```

6) typedef

- **Caution!! typedef and structure definition**
  - Similar but a lot different depending on whether typedef is used or not

```
typedef struct student{
   int id;   char name[8];   double grade;
} STUDENT;        //STUDENT: data type
```

```
struct student{
   int id;   char name[8];   double grade;
} st;             // st: variable
```

6) typedef

- **[Practice 8] Use an array of structures in Practice1. Receive two lunchbox information from a user. Print the information.**
  - Define a user-defined data type for a structure (Practice1) using typedef