

Applications of Structures (1A)

Copyright (c) 2009 - 2017 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

An Array of Structures
Incomplete Definition of Structures
Self-Referential Structures

Defining an array of a structure

Student ID Korean English Math Average

$S[i].I$ $S[i].K$ $S[i].E$ $S[i].M$ $S[i].A$

$S[0]$					
$S[1]$					
$S[2]$					
$S[3]$					
$S[4]$					
$S[5]$					
$S[6]$					
$S[7]$					

```
struct Stype {  
    int    I;  
    int    K;  
    int    E;  
    int    M;  
    double A;  
};  
  
struct Stype S[SIZE];
```

S[i].I, S[i].K, S[i].E, S[i].M, S[i].A

Student ID Korean English Math Average

S[i].I S[i].K S[i].E S[i].M S[i].A



```
struct Stype {  
    int    I;  
    int    K;  
    int    E;  
    int    M;  
    double A;  
};
```

```
struct Stype S[SIZE];
```

S[i].I ≡ ((S[i])).I

Accessing members of an array of a structure

[] , . > *

```
struct Stype {  
    int    I;  
    int    K;  
    int    E;  
    int    M;  
    double A;  
};
```

```
struct Stype S[SIZE];
```

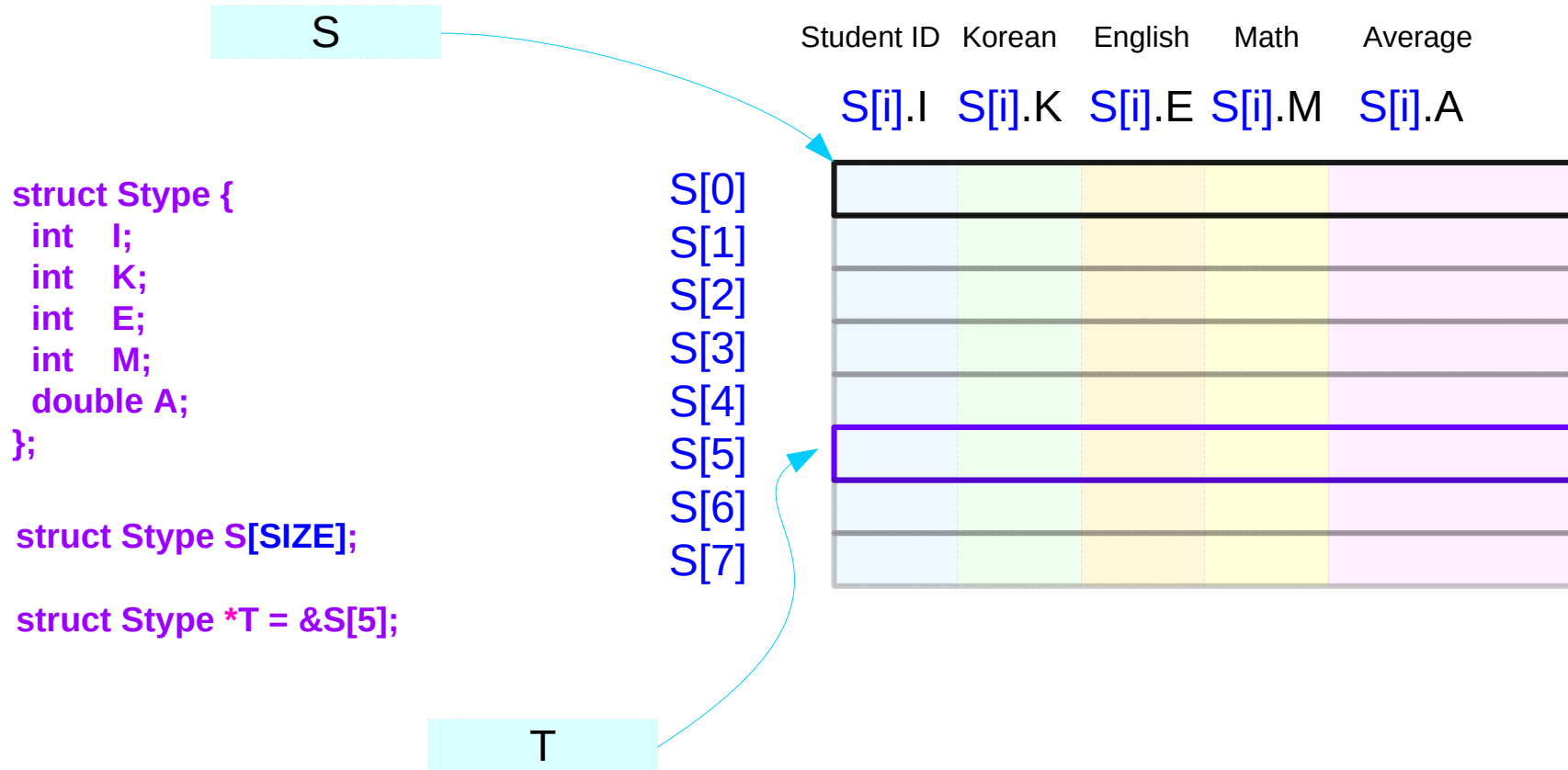
$S[i].I \equiv ((S[i]).I)$

$\equiv (*(S+i)).I$

$\equiv (S+i)->I$

~~$\equiv *(S+i).I$~~ $\equiv *((S+i).I)$

Pointer to a Structure Array



Passing Arrays of Structures

```
struct Stype {  
    int I;  
    int K;  
    int E;  
    int M;  
    double A;  
};
```

```
struct Stype S[SIZE];
```

S



T

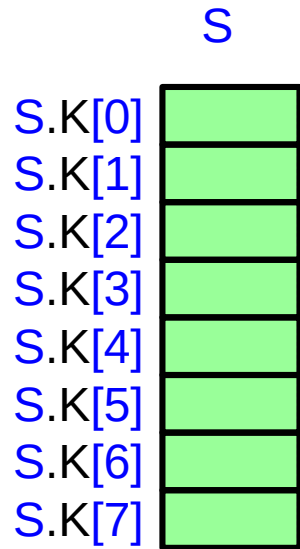
Array Name

Starting Address

Pass by Reference

```
void func (struct Stype T[]);
```


Defining an array of a structure



```
struct Stype {  
    int K[SIZE];  
};
```

```
struct Stype S;
```

```
void func( struct Stype S )
```

Can pass the entire array contents by value

Accessing array members

[] , . > *

$S.K[i] \equiv ((S.K)[i])$

$\equiv *(S.K+i)$

~~$S.(*(K+i))$~~

~~$S.*(K+i)$~~

~~$S.(K+i)$~~

```
struct Stype {  
    int K[SIZE];  
};
```

```
struct Stype S[SIZE];
```

Struct Variable Declaration Summary

```
struct aaa {  
    int    i;  
    short  s;  
    char   c;  
};
```

```
struct aaa var;
```

```
struct aaa {  
    int    i;  
    short  s;  
    char   c;  
};
```

```
typedef struct aaa ATYPE ;
```

```
ATYPE var;
```

```
struct aaa {  
    int    i;  
    short  s;  
    char   c;  
} var ;
```

```
typedef struct aaa {  
    int    i;  
    short  s;  
    char   c;  
} ATYPE ;
```

```
ATYPE var;
```

Incomplete Definition Range

```
struct aaa { int i; short s; char c; };
```

↑ ↑ ↑
At all these points, the structure
type definition is **incomplete**

↑ ↑
From here, the definition complete.

incomplete definition

can be used in cases where the exact **size** information is not required

Dereferencing requires size information

Member assignment require size information

Local variable declaration require size information

Local pointer variable takes always 8bytes on a 64-bit machines

Global variable

Incomplete Definition Usages

```
struct bbb { int i; struct bbb *s; };
```

incomplete definition

can be used with pointer variable declaration

```
typedef struct ccc Ctype;
```

incomplete definition

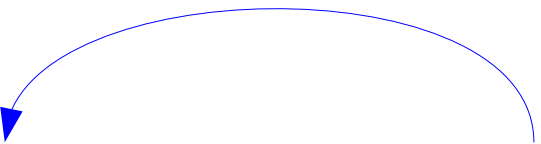
can be used with **typedef**

```
void func( struct ddd *Ctype );
```

incomplete definition

can be used with pointer parameter declaration

Self-Referential Structures

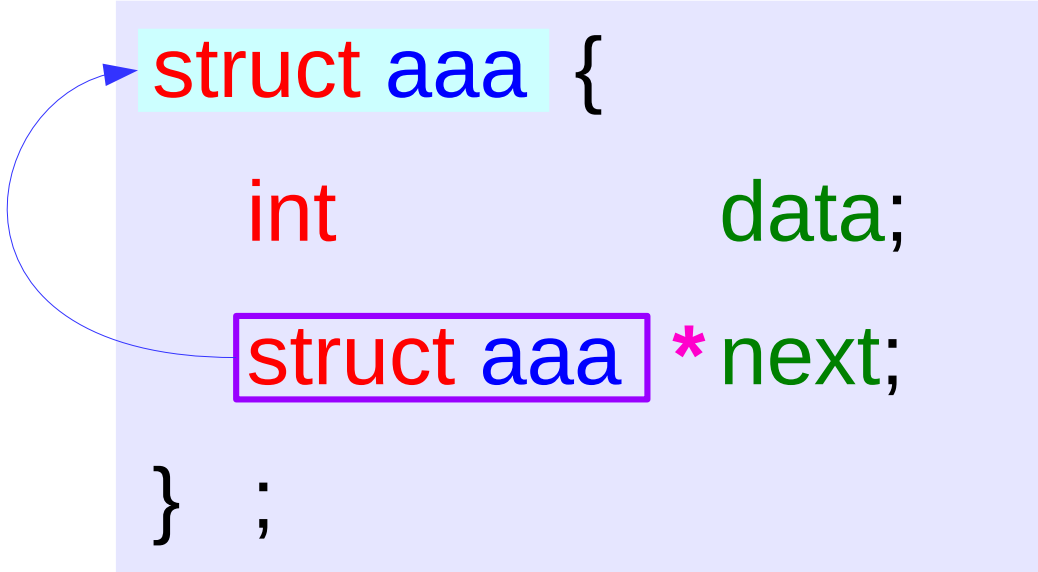


```
struct bbb { int i; struct bbb *s; };
```

incomplete definition
can be used with pointer variable declaration

Self-Referential Structure

```
struct aaa {  
    int data;  
    struct aaa * next;  
};
```



incomplete definition
points to the self type

Self-Referential Structure Definitions

structure type

```
struct aaa {  
    int data;  
    struct aaa *next;  
};
```

```
struct aaa var;
```

structure type

```
struct aaa {  
    int data;  
    struct aaa *next;  
};
```

```
typedef struct aaa ATYPE;
```

```
ATYPE var;
```

structure type

```
struct aaa {  
    int data;  
    struct aaa *next;  
} var;
```

structure type

```
typedef struct aaa {  
    int data;  
    struct aaa *next;  
} ATYPE;
```

```
ATYPE var;
```


Changing Orders : Global Struct Variable Declaration

```
struct aaa var;
```



```
struct aaa {  
    int data;  
    struct aaa *next;  
};
```

```
typedef struct aaa ATYPE;
```

```
ATYPE var;
```



```
struct aaa {  
    int data;  
    struct aaa *next;  
};
```

Global Structure Variable Declaration is possible before the complete structure definition.

Local Structure Variable Declaration is not possible

```
struct aaa {  
    int data;  
    struct aaa *next;  
} var;
```

```
typedef struct aaa {  
    int data;  
    struct aaa *next;  
} ATYPE;
```

```
ATYPE var;
```

Changing Tag Names

```
struct Atype var;
```

```
typedef struct Atype Atype ;
```

```
Atype var;
```

```
struct Atype {  
    int data;  
    struct Atype *next;  
};
```

```
struct Atype {  
    int data;  
    struct Atype *next;  
};
```

aaa → Atype : tag name
Atype : new type

global variable : var

```
struct Atype {  
    int data;  
    struct Atype *next;  
} var;
```

```
typedef struct Atype {  
    int data;  
    struct Atype *next;  
} Atype ;
```

```
Atype var;
```

Self-Referential Data Structures

```
struct node {  
    int data;  
    struct node * next;  
};
```

```
typedef struct node node ;
```

```
struct node {  
    int data;  
    node * next;  
};
```

```
typedef struct node node ;
```

```
typedef struct node node ;
```

```
struct node {  
    int data;  
    struct node * next;  
};
```

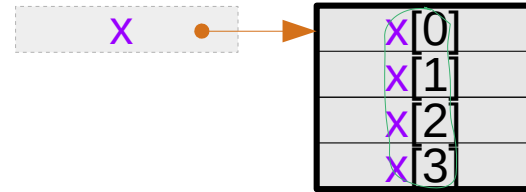
```
typedef struct node node ;
```

```
struct node {  
    int data;  
    node * next;  
};
```

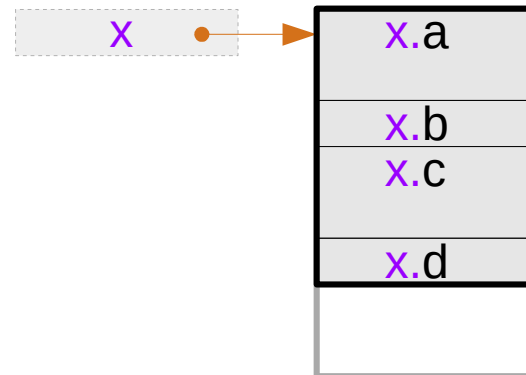
Comparison with a structure type

```
int x[4];
```

A 1-d array



```
struct aaa {  
    long a;  
    int b;  
    long c;  
    int d;  
} x;
```



References

- [1] Essential C, Nick Parlante
- [2] Efficient C Programming, Mark A. Weiss
- [3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
- [4] C Language Express, I. K. Chun
- [5] <https://pdos.csail.mit.edu/6.828/2008/readings/pointers.pdf>