Introduction to Scientific and Engineering Computation (BIL 104E)

Lecture 12

Structures & Unions

Structures

Arrays can be used to collect groups of variables of the same type.

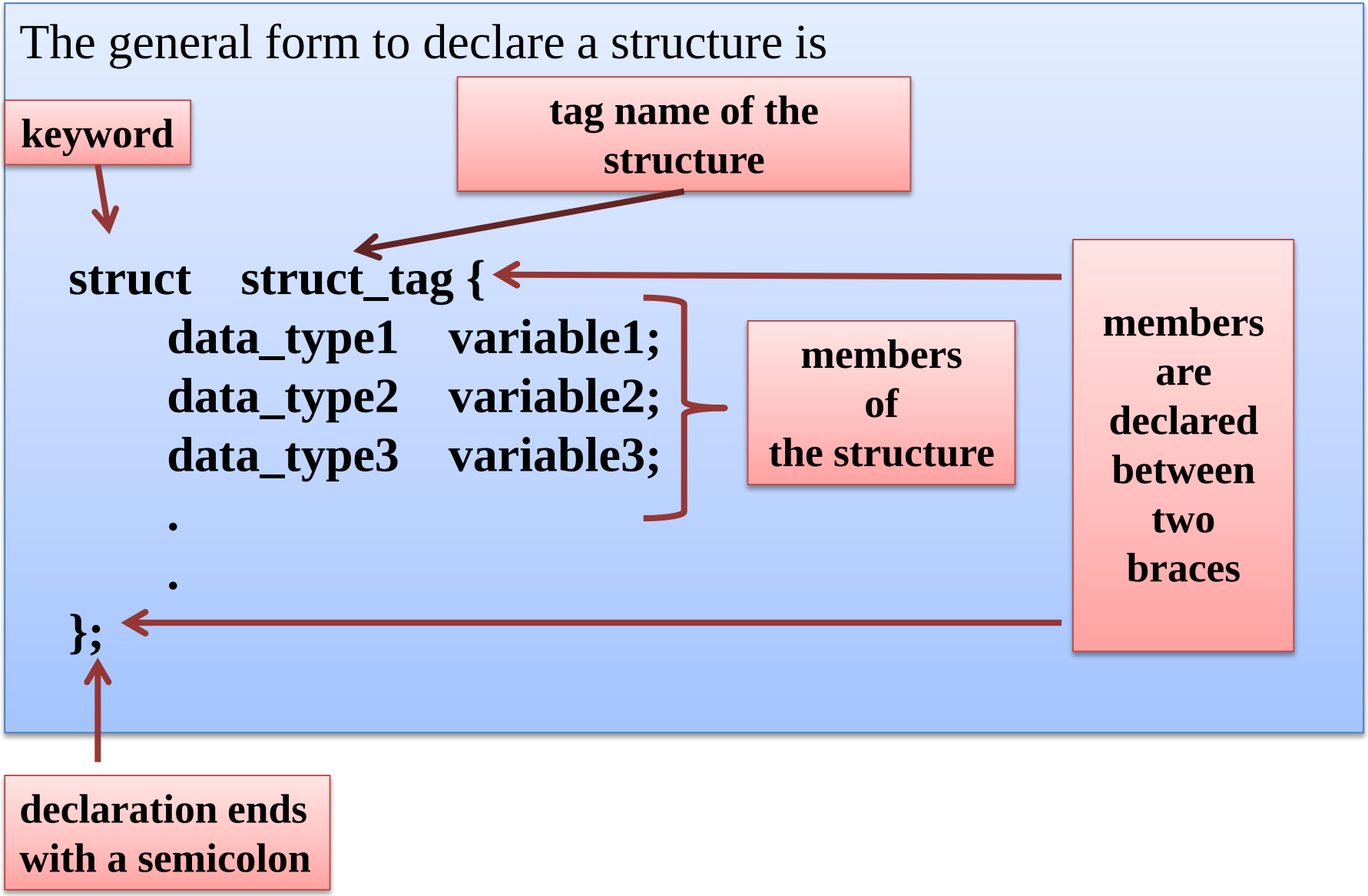
You can group variables of different types with a data type called a **structure**.

A structure collects different data items in such a way that they can be referenced as a single unit.

Data items in a structure which has its own name are called **members of the structure**.

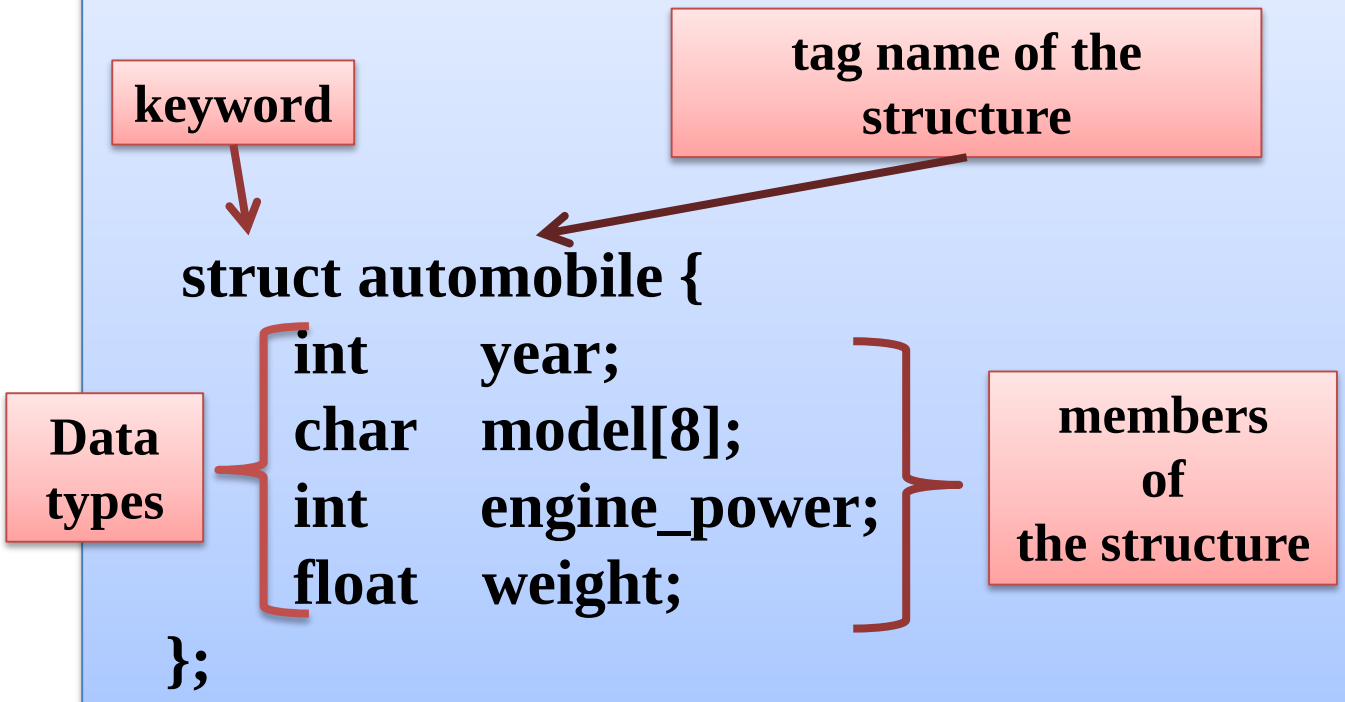
Structures : Declaration

The general form to declare a structure is



Structures : Declaration

For instance,



Structures : Defining Structure Variables

After declaring a structure, you can define the structure variables.

For instance,

struct automobile sedan, pick_up, sport_utility;

Three structure variables, **sedan**, **pick_up**, and **sport_utility**, are defined by the structure of automobile.
All these three structure variables contain the four members of the structure automobile.

You can combine the structure declaration and definition like this:

```
struct automobile {
        int year;
        char model[8];
        int engine_power;
        float weight;
} sedan,  pick_up,  sport_utility;
```

Structures : Referencing Structure Members with the Dot Operator

You can utilize dot operator to reach members of structure variables.

variable_name.member_name is the general form of referencing structure members.

For instance,

sedan.year = 1997;

The integer value 1997 is assigned to the member **year** of structure variable **sedan**.

ptr = sedan.model;

This statement assigns the start address of the character array of **model** to a **char** pointer **ptr**.

Structures : Referencing Structure Members with the Dot Operator

```
#include <stdio.h>
main(void)
{
   struct computer {
         float cost;
         int year;
         int cpu_speed;
         char cpu_type[16];
   } model;
   printf("The type of the CPU inside your computer?\n");
   gets(model.cpu_type);
   printf("The speed(MHz) of the CPU?\n");
   scanf("%d", &model.cpu_speed);
   printf("The year your computer was made?\n");
   scanf("%d", &model.year);
   printf("How much you paid for the computer?($)\n");
   scanf("%f", &model.cost);

   printf("Here are what you entered:\n");
   printf("Year: %d\n", model.year);
   printf("Cost: $%6.2f\n", model.cost);
   printf("CPU type: %s\n", model.cpu_type);
   printf("CPU speed: %d MHz\n", model.cpu_speed);
   getchar();
   getchar();
   return 0;
}
```

Structures : Referencing Structure Members with the Dot Operator

Computer Screen

The type of the CPU inside your computer?
core2
The speed(MHz) of the CPU?
1600
The year your computer was made?
2008
How much you paid for the computer?($)
700
Here are what you entered:
Year: 2008
Cost: $700.00
CPU type: core2
CPU speed: 1600 MHz

Structures : Initializing Structures

```
#include <stdio.h>
main(void)
{
    struct employee {
          int id;
          char name[32];
    };
/* structure initialization */
    struct employee info = {1, "B. Smith"};
    printf("Here is a sample:\n");
    printf("Employee Name: %s\n", info.name);
    printf("Employee ID #: %04d\n\n", info.id);

    printf("What is your name?\n");
    gets(info.name);
    printf("What is your ID number?\n");
    scanf("%d", &info.id);

    printf("\nHere are what you entered:\n");
    printf("Name: %s\n", info.name);
    printf("ID #: %04d\n", info.id);
    getchar();
    getchar();
    return 0;
}
```

Structures : Initializing Structures

Computer Screen

Here is a sample:
Employee Name: B. Smith
Employee ID #: 0001

What is your name?
M. Simsek
What is your ID number?
0002

Here are what you entered:
Name: M. Simsek
ID #: 0002

Structures and Function Calls

In the C language, it is possible to pass an entire structure to a function. In addition, the return value of the function can be a structure.

```
#include <stdio.h>
   struct computer {
         float cost;
         int year;
         int cpu_speed;
         char cpu_type[16];
   };
  typedef struct computer SC; /* create synonym */
 SC DataReceive(SC s); /* function declaration */
main(void)
{
   SC model;
   model = DataReceive(model);
   printf("Here are what you entered:\n");
   printf("Year: %d\n", model.year);
   printf("Cost: $%6.2f\n", model.cost);
   printf("CPU type: %s\n", model.cpu_type);
   printf("CPU speed: %d MHz\n", model.cpu_speed);
```

Structures and Function Calls

```
    getchar();
    getchar();
    return 0;
}
/* function definition */
SC DataReceive(SC s)
{
    printf("The type of the CPU inside your computer?\n");
    gets(s.cpu_type);
    printf("The speed(MHz) of the CPU?\n");
    scanf("%d", &s.cpu_speed);
    printf("The year your computer was made?\n");
    scanf("%d", &s.year);
    printf("How much you paid for the computer?\n");
    scanf("%f", &s.cost);
    return s;
}
```

Structures and Function Calls

Computer Screen

The type of the CPU inside your computer?
core 4
The speed(MHz) of the CPU?
3200
The year your computer was made?
2013
How much you paid for the computer?
1255.35
Here are what you entered:
Year: 2013
Cost: $1255.35
CPU type: core 4
CPU speed: 3200 MHz

Referencing Structures with Pointers

You can declare a structure type pointer and keep the address of the structure type variable inside this pointer.
In addition, it is possible to pass this pointer as an argument to a function.

```
#include <stdio.h>
struct computer {
    float cost;
    int year;
    int cpu_speed;
    char cpu_type[16];
};

typedef struct computer SC;
void DataReceive(SC *ptr_s);
```

Referencing Structures with Pointers

```
main(void)
{
   SC model;
   DataReceive(&model);
   printf("Here are what you entered:\n");
   printf("Year: %d\n", model.year);
   printf("Cost: $%6.2f\n", model.cost);
   printf("CPU type: %s\n", model.cpu_type);
   printf("CPU speed: %d MHz\n", model.cpu_speed);
   getchar();
   getchar();
   return 0;
}
void DataReceive(SC *ptr_s)   /* function definition */
{
   printf("The type of the CPU inside your computer?\n");
   gets((*ptr_s).cpu_type);
   printf("The speed(MHz) of the CPU?\n");
   scanf("%d", &(*ptr_s).cpu_speed);
   printf("The year your computer was made?\n");
   scanf("%d", &(*ptr_s).year);
   printf("How much you paid for the computer?\n");
   scanf("%f", &(*ptr_s).cost);
}
```

Referencing Structures with Pointers

Computer Screen

The type of the CPU inside your computer?
core 4
The speed(MHz) of the CPU?
3500
The year your computer was made?
2013
How much you paid for the computer?
1560
Here are what you entered:
Year: 2013
Cost: $1560.00
CPU type: core 4
CPU speed: 3500 MHz

Referencing a Structure Member with arrow operator ->

The **arrow operator ->** is more frequently used to access **structure members** via pointers, rather than the **dot operator**.

For instance,

```
struct computer {
    float cost;
    int year;
    int cpu_speed;
    char cpu_type[16];
};

struct   computer   variable1;
struct   computer   *ptr_s;
int  *ptr_int;

ptr_s = &variable1;
(*ptr_s).cpu_type  =“core 4”;
ptr_int = &(*ptr_s).cpu_speed;
```

ptr_s -> cpu_type = “core 4”

ptr_int = & (ptr_s -> cpu_speed)

Arrays of Structures

The general form of structure type array is

struct x array_of_structure[8];

Each element of **array_of_structure** is struct type **x.** You can reach each member of structure **x** using **dot operator** or **arrow operator** via struct **x** type pointer.

Arrays of Structures

```
#include<stdio.h>
struct class_list{
      char *name;
      char *sur_name;
      int number;
      int midterm;
      int final;
      int semester;
};
typedef struct class_list student;
main()
{
     student bil104[5];
     student *ptr_s;
     int i;

     ptr_s = bil104;

     ptr_s -> name = "Murat";
     ptr_s -> sur_name = "Simsek";
     ptr_s -> number = 110012094;
     ptr_s -> midterm = 70;
     ptr_s -> final = 80;
     (bil104[1]).semester = ((bil104[1].midterm) + (bil104[1].final))/2 ;
```

Arrays of Structures

```
    (bil104[1]).name = "Hakan";
    (bil104[1]).sur_name = "Kuzu";
    (bil104[1]).number = 110012012;
    (bil104[1]).midterm = 50;
    (bil104[1]).final = 75;
    (bil104[1]).semester = ((ptr_s -> midterm) + (ptr_s -> final))/2 ;

    for (i = 0; i < 2; i++) {
        printf("%d %s  %s  %d\n",ptr_s ->number,ptr_s->name,ptr_s ->sur_name, ptr_s -> semester);
       ++ptr_s;
    }

    getchar();
    return 0;
}
```

Computer Screen

```
110012094    Murat      Simsek    75
110012012    Hakan      Kuzu    62
```

Nested Structures

You can declare a structure member inside the another structure.

For instance,

```
struct   y {
        int i;
        char ch[8];
        struct   x   nested;
};
```

Nested Structures

```
#include<stdio.h>
struct exam_result {
        int quiz;
        int midterm;
        int final;
};
typedef struct exam_result exams;
struct class_list {
        char *name;
        char *surname;
        int number;
        exams bil104;
};
typedef struct class_list C_class;

main()
{
    C_class student[4] = {{"Murat", "Simsek", 110012025, {50, 70, 80} },
                          {"Salih", "Yorulmaz", 110012045, {20, 30, 50}},
                          {"Ugur", "Ince", 110012090, {10, 20, 60}}};

    C_class *ptr_std;
    int i;
    double semester;
```

This syntax is valid only for initialization

Nested Structures

```
    student[3].name = "Servet";
    student[3].surname = "Gok";
    student[3].number = 110009144;
    student[3].bil104.quiz = 0;
    student[3].bil104.midterm = 30;
    student[3].bil104.final = 50;

    ptr_std = student;
    printf("Number   Name   Surname   Semester\n");
    for(i = 0; i < 4; i++){
        semester = ptr_std -> bil104.quiz * 0.3 + ptr_std -> bil104.midterm * 0.4 + ptr_std -> bil104.quiz * 0.4;
        printf("%d %s  %s  %2.2lf\n", ptr_std -> number, ptr_std -> name, ptr_std -> surname, semester  );
        ++ptr_std;
    }

    getchar();
    return 0;
}
```

This syntax is valid only for assignment

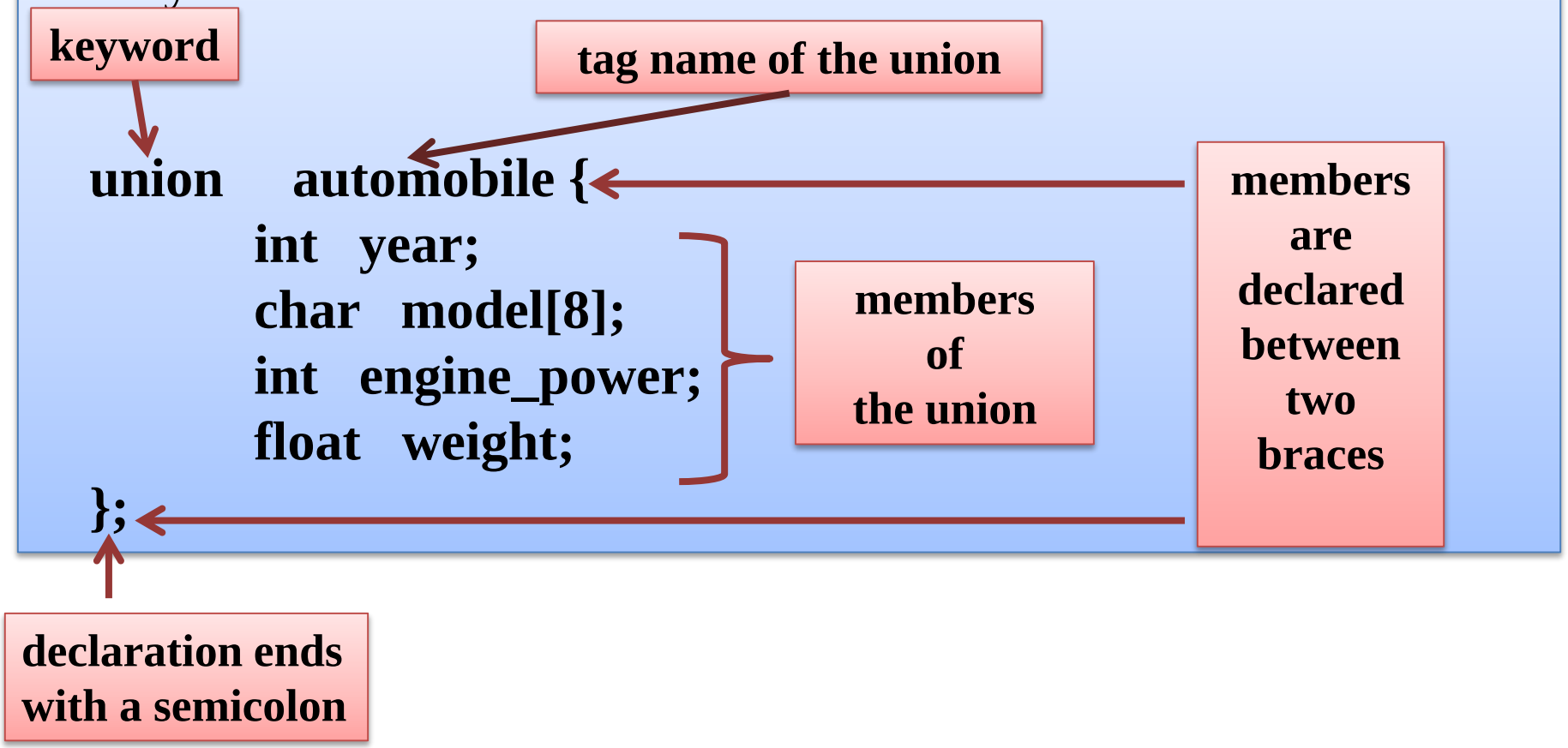
```
Number   Name   Surname   Semester
110012025 Murat  Simsek  63.00
110012045 Salih  Yorulmaz  26.00
110012090 Ugur  Ince  15.00
110009144 Servet  Gok  12.00
```

What Is a Union?

A union is a block of memory that is used to hold data items of different types.

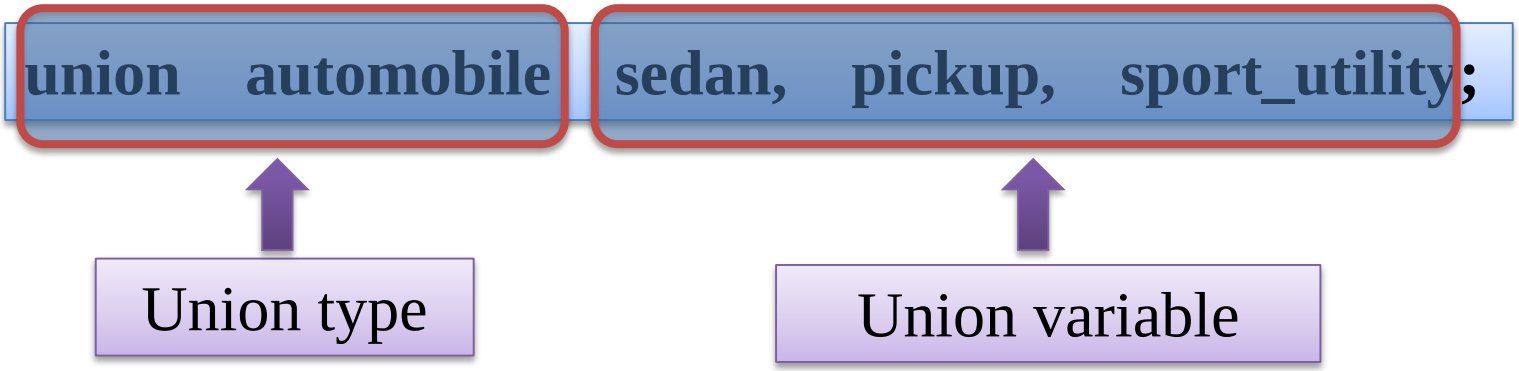
The data items saved in the union are overlaid in order to share the same memory location.

The Syntax of Union Declaration:



Defining Union Variables

You can define union variables after declaring a union.



You can write the previous union declaration and definition together

```
union automobile {
        int year;
        char model[8];
        int engine_power;
        float weight;
} sedan, pickup, sport_utility;
```

Union declaration

Union variable definition

Referencing a Union with . or ->

The **dot operator (**.) can be used in referencing union members.

For instance,

sedan.year = 1997;

In this statement, the value of 1997 is assigned to one of the members of the **sedan union** using **dot operator**.

For instance,

union automobile *ptr;
ptr -> year = 1997;

In this statement, the value of 1997 is assigned to one of the members of the **sedan union** via same type pointer using **arrow operator**.

Initializing a Union

The starting memory location of a union is shared by all the members of the union.

The size of a union is at least as large as the size of the largest data item in the list of the union members.

The value of the latest initialized member overwrites the value of the preceding member.

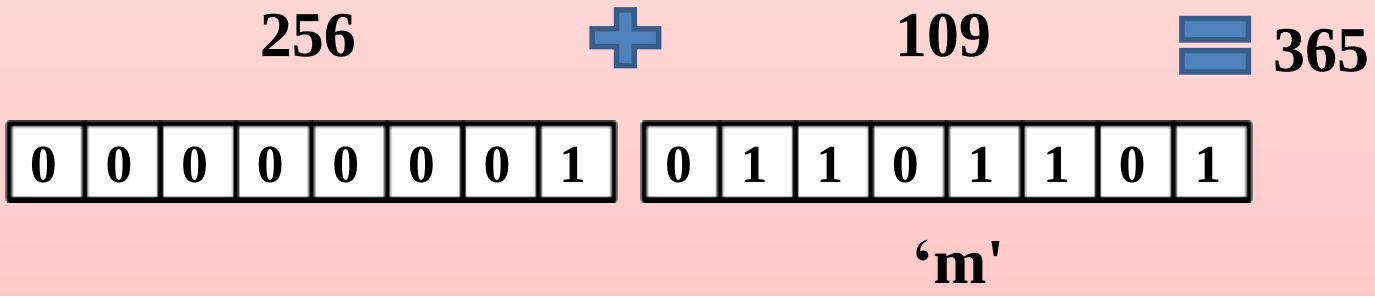
```
main()
{
   union employee {
            int start_year;
            int dpt_code;
            int id_number;
   } info;
   info.year = 1997;
   info.dpt_code = 8;
   info.id_number = 1234;
   printf("%d    %d     %d", info.year , info.dpt_code, info.id_number);
}
```

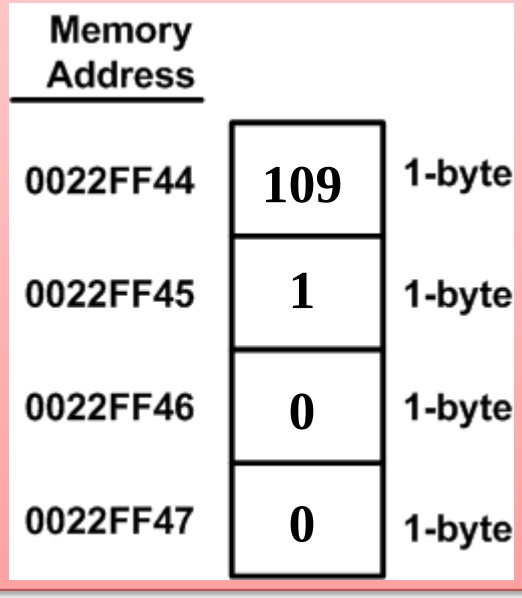
Computer Screen

```
1234    1234    1234
```

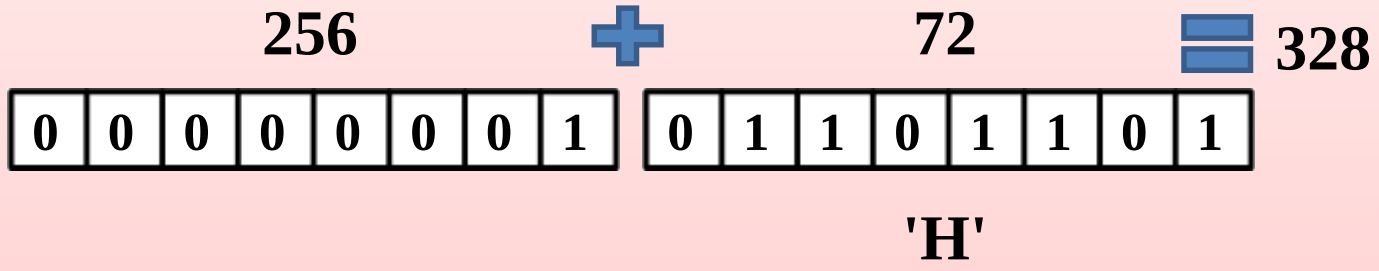
Initializing a Union

```
#include<stdio.h>
main()
{
   union   u {
           char ch;
           int x;
   }  a_union;

   char *ptr_ch;
   int i;

   ptr_ch = &(a_union.ch);
   printf("address of  member ch = %p\n", ptr_ch);
   printf("integer member is changed\n" );
   a_union.x = 365;
   printf("ch = %c\n", a_union.ch );
   printf("x = %d\n", a_union.x );
   for(i = 1; i < 5; i++)
        printf("%d byte = %d\n",i, *(ptr_ch++));
```





Initializing a Union




```
    printf("char member is changed\n");
    a_union.ch = 'H';
    printf("ch = %c\n", a_union.ch );
    printf("x = %d\n", a_union.x );
    ptr_ch = &(a_union.ch);

    for(i = 1; i < 5; i++)
        printf("%d byte = %d\n",i, *(ptr_ch++));

    printf("H = %d\n", 'H');
    printf("m = %d\n", 'm');

    getchar();
    return 0;
}
```

Initializing a Union

Computer Screen

```
address of  member ch = 0022FF44
integer member is changed
ch = m
x = 365
1 byte = 109
2 byte = 1
3 byte = 0
4 byte = 0
char member is changed
ch = H
x = 328
1 byte = 72
2 byte = 1
3 byte = 0
4 byte = 0
H = 72
m = 109
```

The Size of a Union

The members of **a union** all share the same memory location. The size of a **union** is equal to the size of the largest member in the union.

In contrast, each member in a **structure** has its own memory storage. The size of a **structure** is at least equal to the sum of the sizes of its members.

The Size of a Union

```
#include <stdio.h>
#include <string.h>
main(void)
{
   union u {
      double x;
      int y;
   } a_union;

   struct s {
      double x;
      int y;
   } a_struct;

   printf("The size of double: %d-byte\n", sizeof(double));
   printf("The size of int: %d-byte\n", sizeof(int));

   printf("The size of a_union: %d-byte\n", sizeof(a_union));
   printf("The size of a_struct: %d-byte\n", sizeof(a_struct));

   getchar();
   return 0;
}
```

Computer Screen

```
The size of double: 8-byte
The size of int: 4-byte
The size of a_union: 8-byte
The size of a_struct: 16-byte
```

Using Unions

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

struct survey {
   char name[20];
   char c_d_p;
   int age;
   union {
        char cable_company[16];
        char dish_company[16];
   } provider;
};
void DataEnter(struct survey *s);
void DataDisplay(struct survey *s);

main(void)
{
   struct survey tv;
   DataEnter(&tv);
   DataDisplay(&tv);
   getchar();
   getchar();
   return 0;
}
```

Cable_company and dish company are similar and they can be saved same memory area using union.

Using Unions

```
void DataEnter(struct survey *ptr)
{   char is_yes[4];
    printf("Are you using cable at home? (Yes or No)\n");
    gets(is_yes);
    if ((is_yes[0] == 'Y') || (is_yes[0] == 'y'))  {
       printf("Enter the cable company name:\n");
       gets(ptr->provider.cable_company);
       ptr->c_d_p = 'c';
    }
    else {
         printf("Are you using a satellite dish? (Yes or No)\n");
         gets(is_yes);
         if ((is_yes[0] == 'Y') || (is_yes[0] == 'y')){
              printf("Enter the satellite dish company name:\n");
              gets(ptr->provider.dish_company);
              ptr->c_d_p = 'd';
         }
         else
              ptr->c_d_p = 'p';
    }
    printf("Please enter your name:\n");
    gets(ptr->name);
    printf("Your age:\n");
    scanf("%d", &ptr->age);
}
```

Using Unions

```
/* function definition */
void DataDisplay(struct survey *ptr)
{
   printf("\nHere's what you've entered:\n");
   printf("Name: %s\n", ptr->name);
   printf("Age: %d\n", ptr->age);
   if (ptr->c_d_p == 'c')
     printf("Your cable company is: %s\n", ptr->provider.cable_company);
   else
       if (ptr->c_d_p == 'd')
          printf("Your satellite dish company is: %s\n", ptr->provider.dish_company);
       else
          printf("You don't have cable or a satellite dish.\n");
   printf("\nThanks and Bye!\n");
}
```

Using Unions

Computer Screen

```
Are you using cable at home? (Yes or No)
n
Are you using a satellite dish? (Yes or No)
y
Enter the satellite dish company name:
dsmart
Please enter your name:
murat
Your age:
36

Here's what you've entered:
Name: murat
Age: 36
Your satellite dish company is: dsmart

Thanks and Bye!
```

Defining Bit Fields with struct

You can declare a smaller object **a bit field** which allows you to access a single bit. A bit is able to hold only one of two values, 1 or 0.

The general form to declare and define bit fields is

```
struct    tag_name {
        data_type    name1:    length1;
        data_type    name2:    lenght2;
        . . .
        data_type    nameN:    lengthN;
}    variable_list;
```

names of bit fields

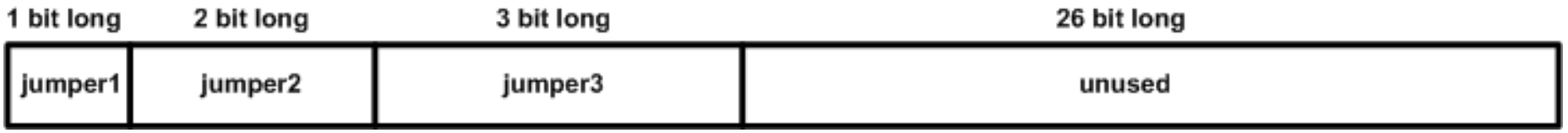
the lengths of bit fields

Defining Bit Fields with struct

For instance, the following statement defines a structure called jumpers with three bit field members:

```
struct   bf {
        int   jumper1:  1;
        int   jumper2:  2;
        int   jumper3:  3;
} jumpers;
```

The memory allocations of jumper1, jumper2, and jumper3 in a 32-bit int.



Defining Bit Fields with struct

```
#include<stdio.h>
struct bit_field {
      int gender: 1; // 1 indicates male 0 indicates female
      int condition: 1; // 1 indicates successful 0 indicates failed
};
typedef struct bit_field bf;
struct class_list {
      char *name;
      bf bit3long;
};
typedef struct class_list class;
main()
{    class bil104[3] = { {"Tolga", 1, 1}, {"Elif", 0, 0} , {"Merve", 0, 1} };
     int i;
     for(i = 1; i < 4; i++) {
          printf("%d   %s  ", i, bil104[i-1].name);
          if(bil104[i-1].bit3long.gender)
              printf("  MALE");
          else
              printf("  FEMALE");
          if(bil104[i-1].bit3long.condition)
              printf("  successful");
          else
              printf("  failed");
          printf("\n");
     }
      getchar();
      return 0;
}
```

Defining Bit Fields with struct

Computer Screen

```
1   Tolga    MALE   successful
2   Elif     FEMALE  failed
3   Merve    FEMALE  successful
```