Structures

Syntax of structure

```
struct structureName
{
dataType member1;
 dataType member2; ...
 };
```

Here is an **example:**

```
struct Person
{
char name[50];
int cityNo;
 float salary;
};
```

Create struct variables

```
struct Person
{
char name[50];
int citNo;
float salary;
};
int main()
{
struct Person person1, person2, p[20];
 return 0;
}
```

```
struct Person
{
char name[50];
int cityNo;
float salary;
 } person1, person2, p[20];
```

ACCESS MEMBERS OF A STRUCTURE

- There are two types of operators used for accessing members of a structure.
- . - Member operator
- -> - Structure pointer operator
- Suppose, you want to access the salary of person2. Here's how you can do it.
- person2.salary

```
// Program to add two distances (feet-inch)
#include <stdio.h>
struct Distance
{
int feet;
float inch;
} dist1, dist2, sum;
int main()
{
printf("1st distance\n");
printf("Enter feet: ");
scanf("%d", &dist1.feet);
printf("Enter inch: ");
scanf("%f", &dist1.inch);
printf("2nd distance\n");
printf("Enter feet: ");
scanf("%d", &dist2.feet);
printf("Enter inch: ");
scanf("%f", &dist2.inch);
// adding feet
sum.feet = dist1.feet + dist2.feet;
// adding inches
sum.inch = dist1.inch + dist2.inch;
// changing to feet if inch is greater than 12
while (sum.inch >= 12)
{
++sum.feet;
sum.inch = sum.inch - 12;
}
printf("Sum of distances = %d\'-%.1f\"", sum.feet,
sum.inch);
return 0;
}
```

ARRAY OF STRUCTURES & ARRAY WITHIN STRUCTURES

```
struct student
{
char name[10] ; \\ array within structure
float percentage ;
int rollno ;
} ;
main( )
{
struct student s[3]; \\ Declaring array of Structure variables;
printf ( "\nEnter names, rollno, & percentage 3 students\n" ) ;
for(int i=0;i<3;i++)
scanf ( "%c %d%f", &s[i].name, &s[i].rollno, &s[i]. percentage ) ;
printf ( "\nAnd this is what you entered" ) ;
for(i=0;i<3;i++)
printf( "%c %d %f", &s[i].name, &s[i].rollno, &s[i]. percentage ) ; }
```

Array of Structures- Another example

```
struct book
   {
    char  name ;
    float  price ;
    int  pages ;
   } ;
 struct  book  b[100] ;

 for ( i = 0 ; i <= 99 ; i++ )
{
   printf ( "\n Enter name, price and pages " ) ;
   scanf ( "%c %f %d", &b[i].name, &b[i].price, &b[i].pages
   ) ;
}
```

Values of One Structure Variable to Another

```
struct employee
{   char  name[10] ;
   int  age ;
   float  salary ;
} ;
struct employee  e1 = { "Sanjay", 30, 5500.50 } ;
struct employee  e2, e3 ;
```

Values of One Structure Variable to Another

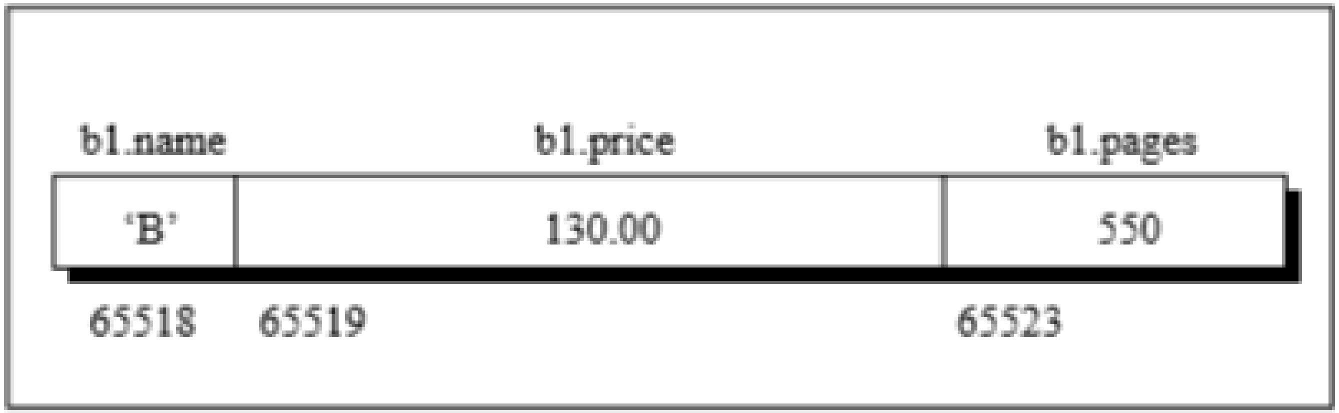
```
/* piece-meal copying */

strcpy ( e2.name, e1.name ) ;

e2.age = e1.age ;

e2.salary = e1.salary ;
```

VALUES OF ONE STRUCTURE VARIABLE TO ANOTHER

```
/* copying all elements at one go */

e3 = e2 ;

printf ( "\n%s %d %f", e1.name, e1.age, e1.salary ) ;

printf ( "\n%s %d %f", e2.name, e2.age, e2.salary ) ;

printf ( "\n%s %d %f", e3.name, e3.age, e3.salary ) ; }
```

How Structure Elements are Stored

```
struct book
{
   char  name ;
   float  price ;
   int  pages ;
} ;
struct book  b1 = { 'B', 130.00, 550 } ;
```



KEYWORD TYPEDEF

- We use the typedef keyword to create an alias name for data types. It is commonly used with structures to simplify the syntax of declaring variables.

This code

```
struct Distance
{
int feet;
float inch;
};
 int main()
{
 structure Distance d1, d2;
}
```

is equivalent to

```
typedef struct Distance
{
int feet;
float inch;
} distances;
int main()
{
distances d1, d2;
 }
```

Nested Structures

```
struct complex
 {
int imag;
float real;
};
struct number
 {
struct complex comp;
int integers;
} num1, num2;
```

C Pointers to struct

```
struct name {
member1;
member2;
.
.
};
int main()
{
struct name *ptr, Harry;
}
```

Here, ptr is a pointer to struct.

Access members using Pointer

```
#include <stdio.h>
struct person
{
int age;
float weight;
};
int main()
{
struct person *personPtr, person1;
personPtr = &person1;
printf("Enter age: ");
scanf("%d", &personPtr->age);
printf("Enter weight: ");
scanf("%f", &personPtr->weight);
printf("Displaying:\n");
printf("Age: %d\n", personPtr->age);
printf("weight: %f", personPtr->weight);
return 0;
}
```

- personPtr->age is equivalent to (*personPtr).age
- personPtr->weight is equivalent to (*personPtr).weight

Dynamic memory allocation of structs

```
#include <stdio.h>
#include <stdlib.h>
struct person {
int age;
char name[30];
};
int main()
{
struct person *ptr;
int i, n;
printf("Enter the number of persons: ");
scanf("%d", &n);
// allocating memory for n numbers of struct person
ptr = (struct person*) malloc(n * sizeof(struct person));
for(i = 0; i < n; ++i)
{
printf("Enter first name and age respectively: ");
// To access members of 1st struct person,
// ptr->name and ptr->age is used
// To access members of 2nd struct person,
// (ptr+1)->name and (ptr+1)->age is used
scanf("%s %d", (ptr+i)->name, &(ptr+i)->age);
}
printf("Displaying Information:\n");
for(i = 0; i < n; ++i)
printf("Name: %s\tAge: %d\n", (ptr+i)->name, (ptr+i)
->age);
return 0;
}
```

Passing Struct to Functions

```
#include <stdio.h>
struct student
{
char name[50];
int age;
};
// function prototype
void display(struct student s);
int main()
{
struct student s1;
printf("Enter name: ");
scanf("%s", s1.name);
printf("Enter age: ");
scanf("%d", &s1.age);
display(s1); // passing struct as an argument
return 0;
}
void display(struct student s)
{
printf("\nDisplaying information\n");
printf("Name: %s", s.name);
printf("\nAge: %d", s.age);
}
```

Return struct from a function

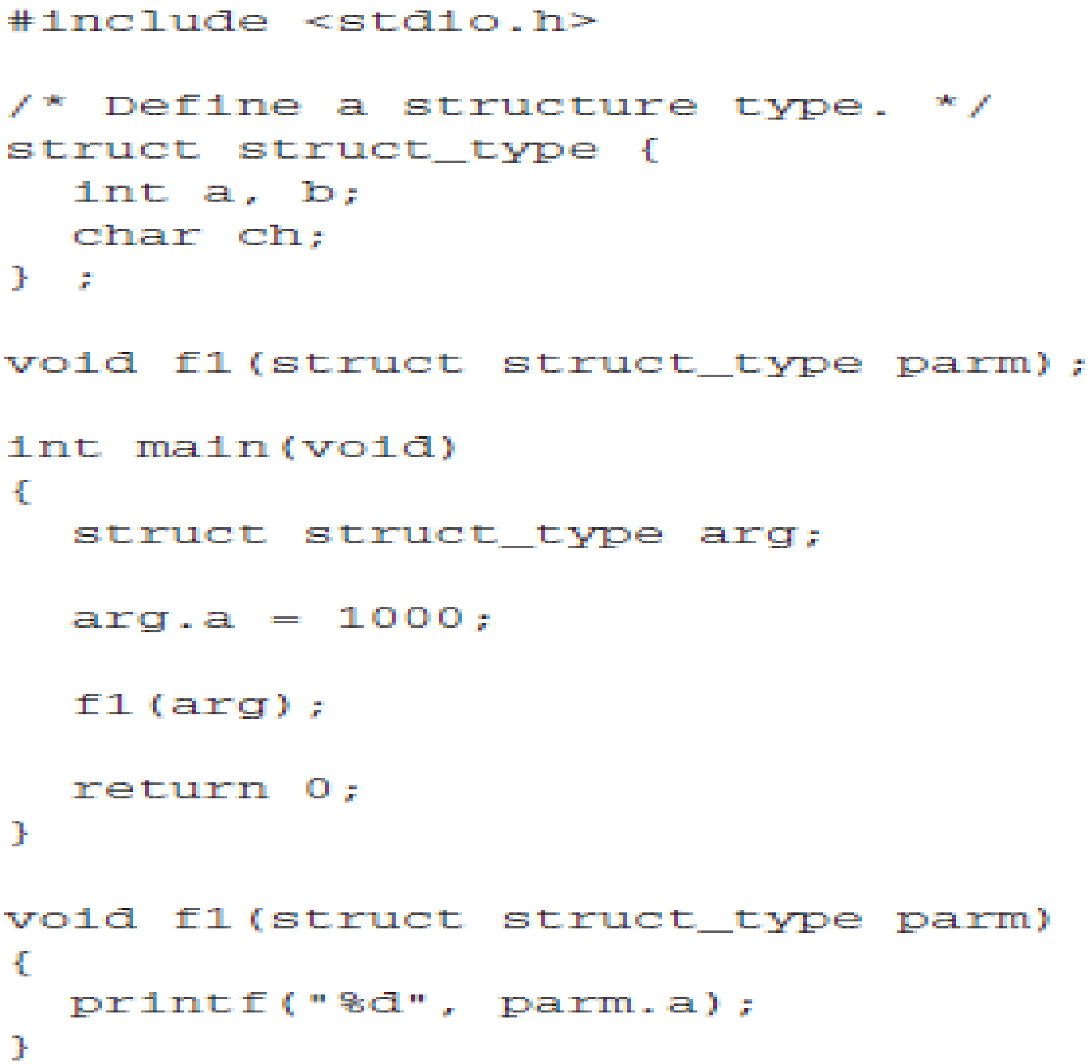
```
#include <stdio.h>
struct student
{
char name[50];
int age;
};
// function prototype
struct student getInformation();
int main()
{
struct student s;
s = getInformation();
printf("\nDisplaying information\n");
printf("Name: %s", s.name);
printf("\nRoll: %d", s.age);
return 0;
}
```

```
struct student getInformation()
{
struct student s1;
printf("Enter name: ");
scanf %s", s1.name);
printf("Enter age: ");
scanf("%d", &s1.age);
return s1;
}
```

Passing struct by reference

```
#include <stdio.h>
typedef struct Complex
{
float real;
float imag;
} complex;
void addNumbers(complex c1, complex c2, complex *result);
int main()
{
complex c1, c2, result;
printf("For first number,\n");
printf("Enter real part: ");
scanf("%f", &c1.real);
printf("Enter imaginary part: ");
scanf("%f", &c1.imag);
printf("For second number, \n");
printf("Enter real part: ");
scanf("%f", &c2.real);
printf("Enter imaginary part: ");
scanf("%f", &c2.imag);
addNumbers(c1, c2, &result);
printf("\nresult.real = %.1f\n", result.real);
printf("result.imag = %.1f", result.imag);
return 0;
}
void addNumbers(complex c1, complex c2, complex *result)
{
result->real = c1.real + c2.real;
result->imag = c1.imag + c2.imag;
}
```

Passing entire structure to functions



```
#include <stdio.h>

/* Define a structure type. */
struct struct_type {
  int a, b;
  char ch;
} ;

void f1(struct struct_type parm);

int main(void)
{
  struct struct_type arg;

  arg.a = 1000;

  f1(arg);

  return 0;
}

void f1(struct struct_type parm)
{
  printf("%d", parm.a);
}
```

Unions

DEFINITION

- A *union* is a memory location that is shared by two or more different types of variables.
- A union provides a way of interpreting the same bit pattern in two or more different ways.

```
union car
{
char name[50];
int price;
};
```

Memory sharing

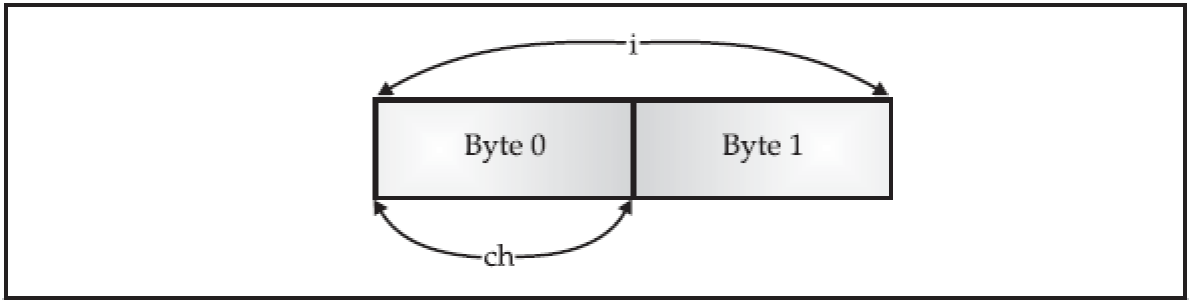


Figure 7-2. *How **i** and **ch** utilize the union **cnvt** (assume 2-byte integers)*

Creating Union Variables

```
union car
{
char name[50];
int price;
};
int main()
{
union car car1, car2, *car3;
return 0;
}
```

Form of Declaration

```
union car
{
char name[50];
int price;
} car1, car2, *car3;
```

Access members of a union

- To access price for car1, car1.price is used.
- To access price using car3, either (*car3).price or car3->price can be used.

Difference between Structures and Union

```
#include <stdio.h>
union unionJob
{
//defining a union
char name[32];
float salary;
int workerNo;
} uJob;
```

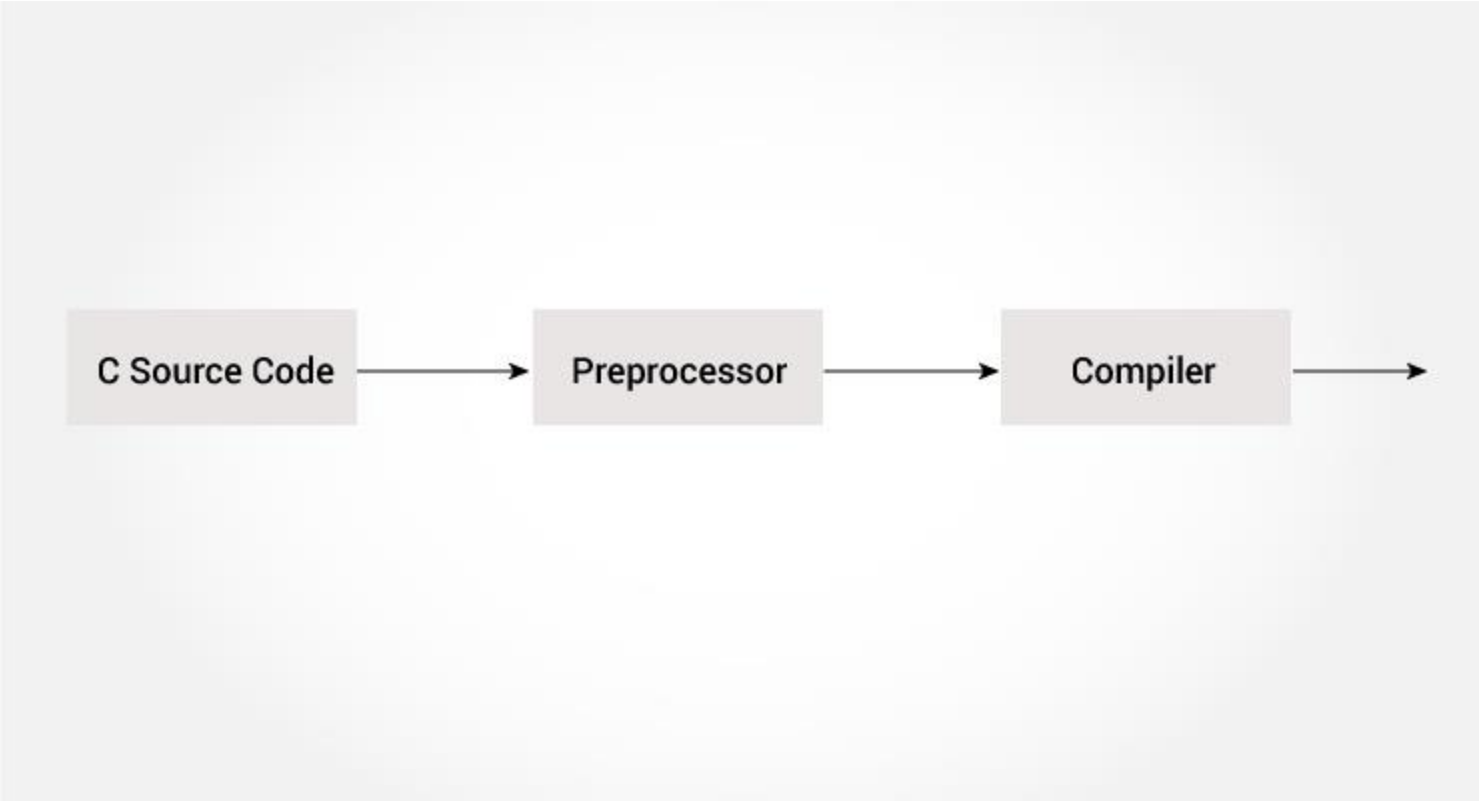
```
struct structJob
{
char name[32];
float salary;
int workerNo;
} sJob;
int main()
{
printf("size of union = %d bytes", sizeof(uJob));
printf("\nsize of structure = %d bytes", sizeof(sJob));
return 0;
}
```

```
#include <stdio.h>
union Job
{
float salary;
int workerNo;
} j;
```

Member access

```
int main()
{
j.salary = 12.3;
j.workerNo = 100;
printf("Salary = %.1f\n", j.salary);
printf("Number of workers = %d", j.workerNo);
return 0;
}
```

Macros in C



- The C preprocessor is a macro preprocessor (allows you to define macros) that transforms your program before it is compiled. These transformations can be the inclusion of header file, macro expansions etc.
- All preprocessing directives begin with a # symbol. For example,
- #define PI 3.14

INCLUDING HEADER FILES: #INCLUDE

- The #include preprocessor is used to include header files to C programs. For example,
- #include <stdio.h>
- Here, stdio.h is a header file. The #include preprocessor directive replaces the above line with the contents of stdio.h header file.
- That's the reason why you need to use #include <stdio.h> before you can use functions like scanf() and printf().
- You can also create your own header file containing function declaration and include it in your program using this preprocessor directive.
- #include "my_header.h"

Macros using #define

- #define c 299792458

#DEFINE PREPROCESSOR

```
#include <stdio.h>
#define PI 3.1415
int main()
{
float radius, area;
printf("Enter the radius: ");
scanf("%f", &radius);
// Notice, the use of PI
area = PI*radius*radius;
printf("Area=%.2f",area);
return 0;
}
```

Function like Macros

- #define circleArea(r) (3.1415*(r)*(r))

```
#include <stdio.h>
#define PI 3.1415
#define circleArea(r) (PI*r*r)
int main() {
float radius, area;
printf("Enter the radius: ");
scanf("%f", &radius);
area = circleArea(radius);
printf("Area = %.2f", area);
return 0;
}
```

How to use conditional?

- **#ifdef Directive**
- #ifdef MACRO
- // conditional codes
- #endif

- #if expression
- // conditional codes
- #endif
- Here, expression is an expression of integer type (can be integers, characters, arithmetic expression, macros and so on).
- The conditional codes are included in the program only if the expression is evaluated to a non-zero value.

- The optional #else directive can be used with #if directive.
- #if expression
- conditional codes if expression is non-zero
- #else
- conditional if expression is 0
- #endif

- #if expression
- // conditional codes if expression is non-zero
- #elif expression1
- // conditional codes if expression is non-zero
- #elif expression2
- // conditional codes if expression is non-zero
- #else
- // conditional if all expressions are 0
- #endif

TO CHECK MACRO

- The special operator #defined is used to test whether a certain macro is defined or not. It's often used with #if directive.
- #if defined BUFFER_SIZE && BUFFER_SIZE >= 2048
- // codes

Macro	Value
__DATE__	A string containing the current date
__FILE__	A string containing the file name
__LINE__	An integer representing the current line number
__STDC__	If follows ANSI standard C, then the value is a nonzero integer
__TIME__	A string containing the current time.

```
#include <stdio.h>
int main()
{
printf("Current time: %s",__TIME__);
}
```