

Structures or aggregates and defination

- Collection of related variables under one name.
- It may have variables of different datatypes.
- We use structures, commonly to define records to be stored in files.

```
#include <stdio.h>
struct comp{
    float re;
    float im;
};
int main(){
    struct comp a, *b;
    a.re = 3; a.im = 4; b = &a;
    printf("From varialbe: %f + i %f\n", a.re, a.im);
    printf("From pointer: %f + i %f\n", b->re, b->im);
    return 0;
}
```

For a normal variable, we use dot operator, also known as **structure member operator (.)**

And **structure pointer operator** or the arrow operator (->), in which minus (-) is followed by greater than (>) is used for structure.

Structure member and structure pointer operators

```
struct Books book1, book2;  
strcpy(book1.title, "Concepts of Programming Languages");  
strcpy(book1.author, "Robert W. Sebesta");  
strcpy(book1.subject, "Programming languages");  
book1.book_id = 131395319;
```

/* specification of book1 */

```
strcpy(book2.title, "C how to program");  
strcpy(book2.author, "Deitel - Deitel");  
strcpy(book2.subject, "C programming language");  
book2.book_id = 136123562;
```

/* specification of book2 */

```
printf( "book 1 title : %s\n", book1.title);  
printf( "book 1 author : %s\n", book1.author);  
printf( "book 1 subject : %s\n", book1.subject);  
printf( "book 1 book_id : %d\n", book1.book_id);
```

printing information
of book1

```
printf( "book 2 title : %s\n", book2.title);  
printf( "book 2 author : %s\n", book2.author);  
printf( "book 2 subject : %s\n", book2.subject);  
printf( "book 2 book_id : %d\n", book2.book_id);
```

printing information
of book2

```
struct Books{  
    char title[50];  
    char author[50];  
    char subject[50];  
    int book_id;  
};
```

An example of Union

Unions are derived data types like structures, but with members that share the same storage space.

```
union number{
    int x;
    double y;
};
union mydata{
    int i;
    float f;
    char s[20];
};
int main(){
    union number value;
    value.x = 100;
    printf("%s\n%s\n %d\n%s\n %f\n\n",
        "Put a value in the integer member and print both members.",
        "int:", value.x, "double: ", value.y);
    value.y=100;
    printf("%s\n%s\n %d\n%s\n %f\n\n",
        "Put a value in the floating member and print both members. "
```

```
value.y=100;
printf("%s\n%s\n %d\n%s\n %f\n\n",
    "Put a value in the floating member and print both members.",
    "int:", value.x, "double: ", value.y);
```

```
union mydata data;
data.i=20;
data.f= 1.729;
strcpy(data.s, "C Programming");
printf("It will print only last value correctly\n%s%d\n%s%f\n%s%s\n",
    "data.i = ",data.i, "data.f = ",data.f,"data.s = ",data.s);
```

```
data.i=10;
printf("data.i = %d\n",data.i);

data.f= 17.29;
printf("data.f = %f\n",data.f);
strcpy(data.s, "It is C Programming");
printf("data.s = %s\n",data.s);
return 0;
```

```
}
```

```
union number{
    int x;
    double y;
};
union mydata{
    int i;
    float f;
    char s[20];
};
```

Enumeration type use of class

An enumeration (enum) is a set of integer enumeration constants represented by identifiers.

```
enum months{JAN=1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };
int main(){
    enum months m;
    const char *mName[] = {"", "January", "February", "March", "April",
                           "May", "June", "July", "August", "September",
                           "October", "November", "December"};
    for(m=JAN; m<=DEC; m++){
        printf("%2d%11s\n", m, mName[m]);
    }
    return 0;
}
```

Rewriting slide 1 with function call

```
void printbook(int, struct Books *);  
int main(){  
    struct Books book1, book2;  
    /* specification of book1 */  
    strcpy(book1.title, "Concepts of Programming Languages");  
    strcpy(book1.author, "Robert W. Sebesta");  
    strcpy(book1.subject, "Programming languages");  
    book1.book_id = 131395319;  
  
    /* specification of book2 */  
    strcpy(book2.title, "C how to program");  
    strcpy(book2.author, "Deitel - Deitel");  
    strcpy(book2.subject, "C programming language");  
    book2.book_id = 136123562;  
  
    /* printing book1 */  
    printbook(1, &book1);  
  
    /* printing book2 */  
    printbook(2, &book2);  
    return 0;  
}
```

```
void printbook(int i, struct Books *book){  
    printf( "book %d title : %s\n", i, book->title);  
    printf( "book %d author : %s\n", i, book->author);  
    printf( "book %d subject : %s\n", i, book->subject);  
    printf( "book %d book_id : %d\n", i, book->book_id);  
}
```

Creating synonyms (or aliases)

```
struct {  
    unsigned int age : 3; /* defines memory bits */  
} aayu;  
  
int main( ) {  
  
    aayu.age = 4;  
    printf( "Sizeof( aayu ) : %lu\n", sizeof(aayu) );  
    printf( "aayu.age : %d\n", aayu.age );  
  
    aayu.age = 7;  
    printf( "aayu.age : %d\n", aayu.age );  
  
    aayu.age = 8; /* It is not in memory size */  
    printf( "aayu.age : %d\n", aayu.age );  
  
    return 0;  
}
```

Creating synonyms using structure pointer operators

```
typedef struct Books{
    char title[50];
    char author[50];
    char subject[50];
    int book_id;
}Book;
int main(){
    Book book;

    strcpy(book.title, "C how to program");
    strcpy(book.author, "Deitel - Deitel");
    strcpy(book.subject, "C programming language");
    book.book_id = 136123562;

    printf( "book title : %s\n", book.title);
    printf( "book author : %s\n", book.author);
    printf( "book subject : %s\n", book.subject);
    printf( "book book_id : %d\n", book.book_id);
    return 0;
}
```