Structures and Unions

Unit 6 Objectives

- **Use structures and unions to store data of different types under one identifier name**

- **Declare, initialize, and use**
  - **Structures**
  - **Arrays of structures**
  - **Pointers to structures**

- **Declare, initialize, and use**
  - **Unions**
  - **Arrays of unions**
  - **Pointers to unions**

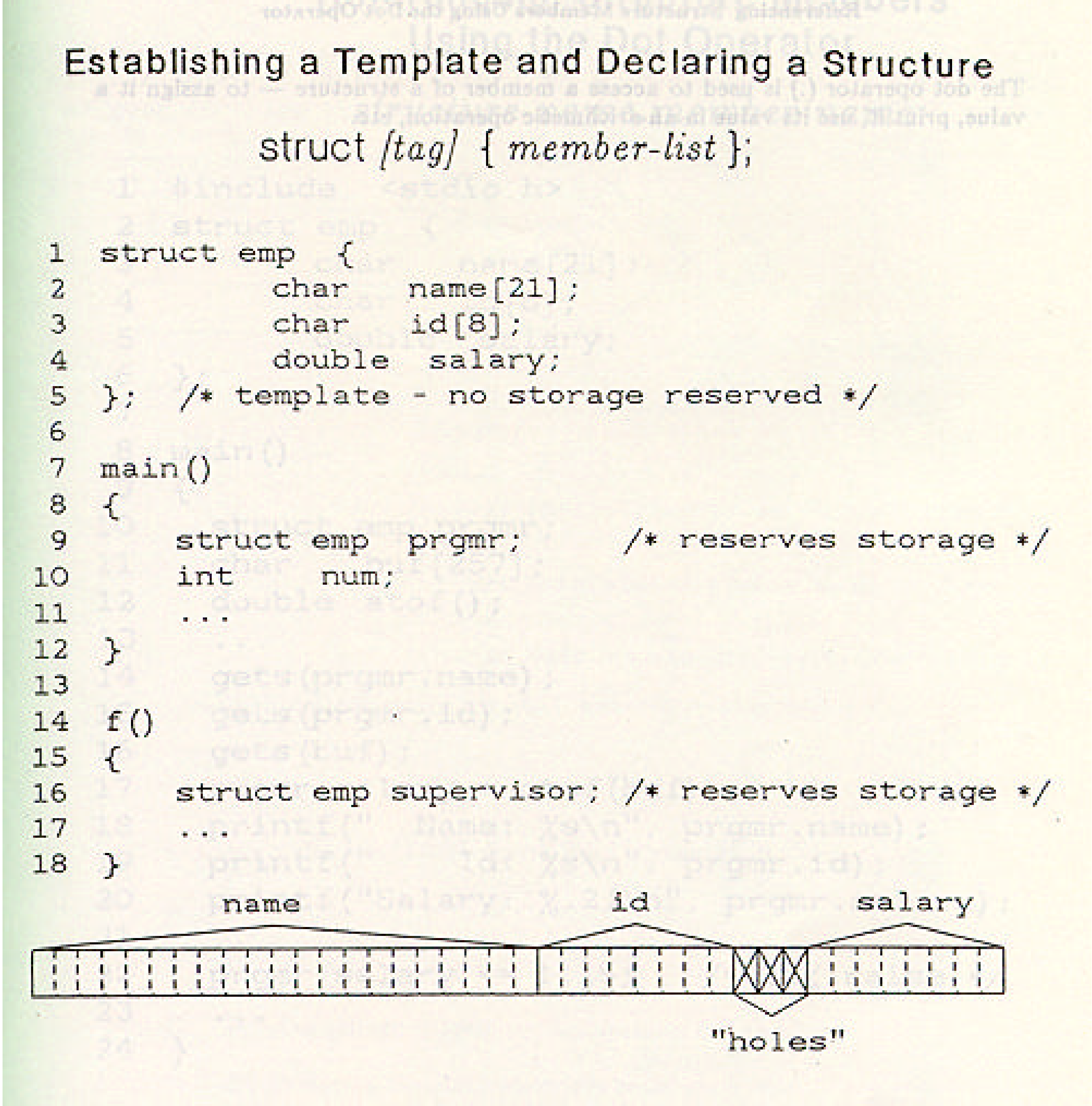
- **Understand the similarities and differences between structures and unions**

Structures

Structures Overview

- What is a structure?
  - A "record"
  - A collection of one or more variables, possibly of different types, grouped under a single name.
  - A derived data type
- Advantages of structures?
  - Used to organize data
  - Allows a group of related variables to be treated as a unit.

name	id	salary



Establishing a Template and Declaring a Structure

struct *[tag]* { *member-list* };

```
struct emp  {
        char   name[21];
        char   id[8];
        double  salary;
};  /* template - no storage reserved */

main()
{
    struct emp  prgmr;      /* reserves storage */
    int    num;
    ...
}

f()
{
    struct emp supervisor; /* reserves storage */
    ...
}
```

name
id
salary
"holes"

Referencing Structure Members Using the Dot Operator

structure-name.member-name

```
#include  <stdio.h>
struct emp  {
        char        name[21] ;
        char        id[8] ;
        double      salary ;
} ;

main()
{
        struct      emp         prgmr ;
        char        buf[257] ;
        double      atof() ;
        • • •
        gets(prgmr.name) ;
        gets(prgmr.id) ;
        gets(buf) ;
        prgmr.salary = atof(buf) ;
        printf(“  Name :  %s\n”, prgmr.name) ;
        printf(“        Id :  %s\n”, prgmr.id) ;
        printf(“ Salary :  %.2f\n”, prgmr.salary) ;
         • • •
        prgmr.salary *= 1.15 ;          /* 15% raise */
        • • •
}
```

Exercise - Structures

Using the program below answer the questions that follow.

```
/* Program to track ticket sales */
#include  <stdio.h>

struct t_info {
      ↑
      Ⓐ    float  price;      Ⓑ          Ⓒ
            int    sold;
};

main()
{
        struct t_info   ticket;   Ⓓ
        int    num;

    Ⓔ                            /* Initializes price to 6 dollars */

    Ⓕ                            /* Initializes number sold to 0 */

    Ⓖ                            /* Adds 5 to number of tickets sold */

        /* Line below prints total amount collected * /

    Ⓗ   printf("%.2f\n",
}
```

1. Connect A through D with descriptions P through S:

   A   P - Members or fields of **struct t_info**

   B   Q - Declares and reserves storage for one structure

   C   R - Tag, allows subsequent declarations of this type

   D   S - Structure template, describes members of the **struct t_info** type, does not reserve storage

2. Fill in the statements labeled E, F, G, and H.

3. The data type of num is **int**. What is **ticket**'s data type?

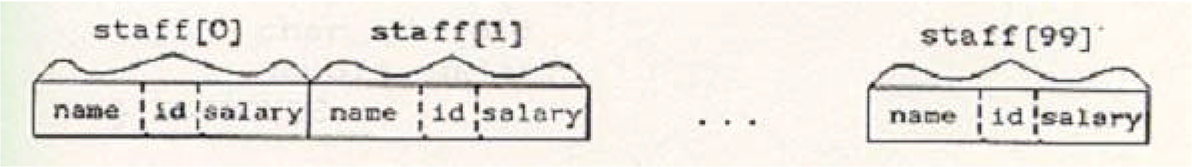
Array of Structures

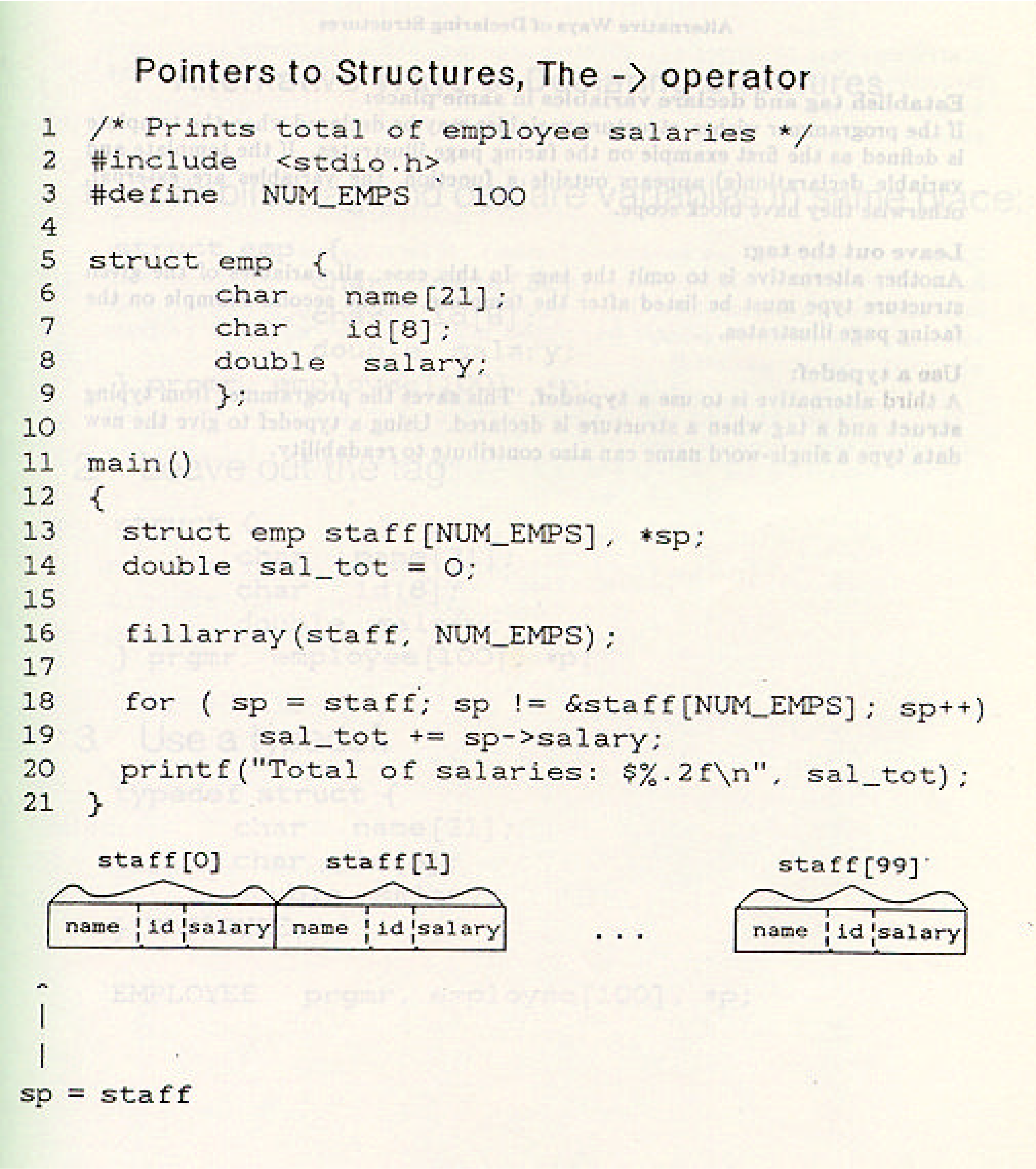
```
/* Prints total of employee salaries */
#include        <stdio.h>
#define         NUM_EMPS        100

struct   emp   {
                char        name[21] ;
                char        id[8] ;
                double      salary ;
} ;

main()
{
      struct  emp  staff[NUM_EMPS] ;
      int i ;
      double  sal_tot = 0 ;

      fillarray(staff, NUM_EMPS) ;
      for (i = 0 ; i < NUM_EMPS ; i ++)
                sal_tot += staff[i].salary ;
      printf("Total of salaries: $%.2f\n", sal_tot) ;
}
```





Pointers to Structures, The -> operator

```
/* Prints total of employee salaries */
#include  <stdio.h>
#define   NUM_EMPS   100

struct emp  {
        char   name[21];
        char   id[8];
        double  salary;
        };

main()
{
  struct emp staff[NUM_EMPS], *sp;
  double  sal_tot = 0;

  fillarray(staff, NUM_EMPS);

  for ( sp = staff; sp != &staff[NUM_EMPS]; sp++)
          sal_tot += sp->salary;
  printf("Total of salaries: $%.2f\n", sal_tot);
}
```

staff[0]
name
id
salary
staff[1]
name
id
salary
. . .
staff[99]
name
id
salary
sp = staff

Alternative Ways of Declaring Structures

1. Establish tag and declare variables in same place :

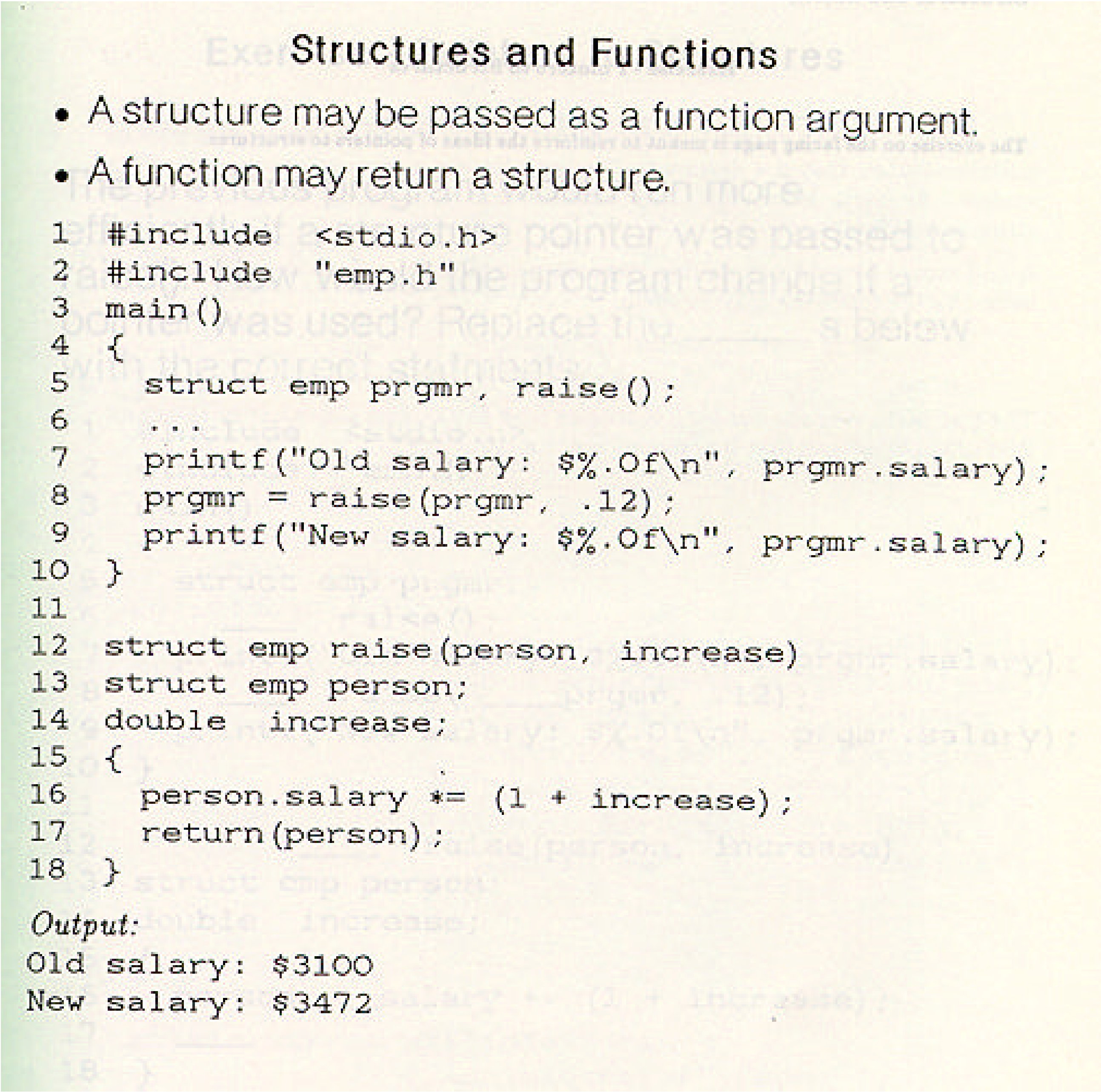
```
struct  emp  {
                    char      name[21] ;
                    char      id[8] ;
                    double    salary ;
} prgmr, employee[100], *p ;
```

2. Leave out the tag :

```
struct  {
        char      name[21] ;
        char      id[8] ;
        double    salary ;
} prgmr, employee[100], *p ;
```

3. Use a typedef :

```
typedef  struct  {
        char      name[21] ;
        char      id[8] ;
        double    salary ;
} EMPLOYEE ;
EMPLOYEE  prgmr,  employee[100], *p ;
```



Structures and Functions

- A structure may be passed as a function argument.
- A function may return a structure.

```
#include  <stdio.h>
#include  "emp.h"
main()
{
  struct emp prgmr, raise();
  ...
  printf("Old salary: $%.0f\n", prgmr.salary);
  prgmr = raise(prgmr, .12);
  printf("New salary: $%.0f\n", prgmr.salary);
}

struct emp raise(person, increase)
struct emp person;
double  increase;
{
  person.salary *= (1 + increase);
  return(person);
}
```

Output:

```
Old salary: $3100
New salary: $3472
```

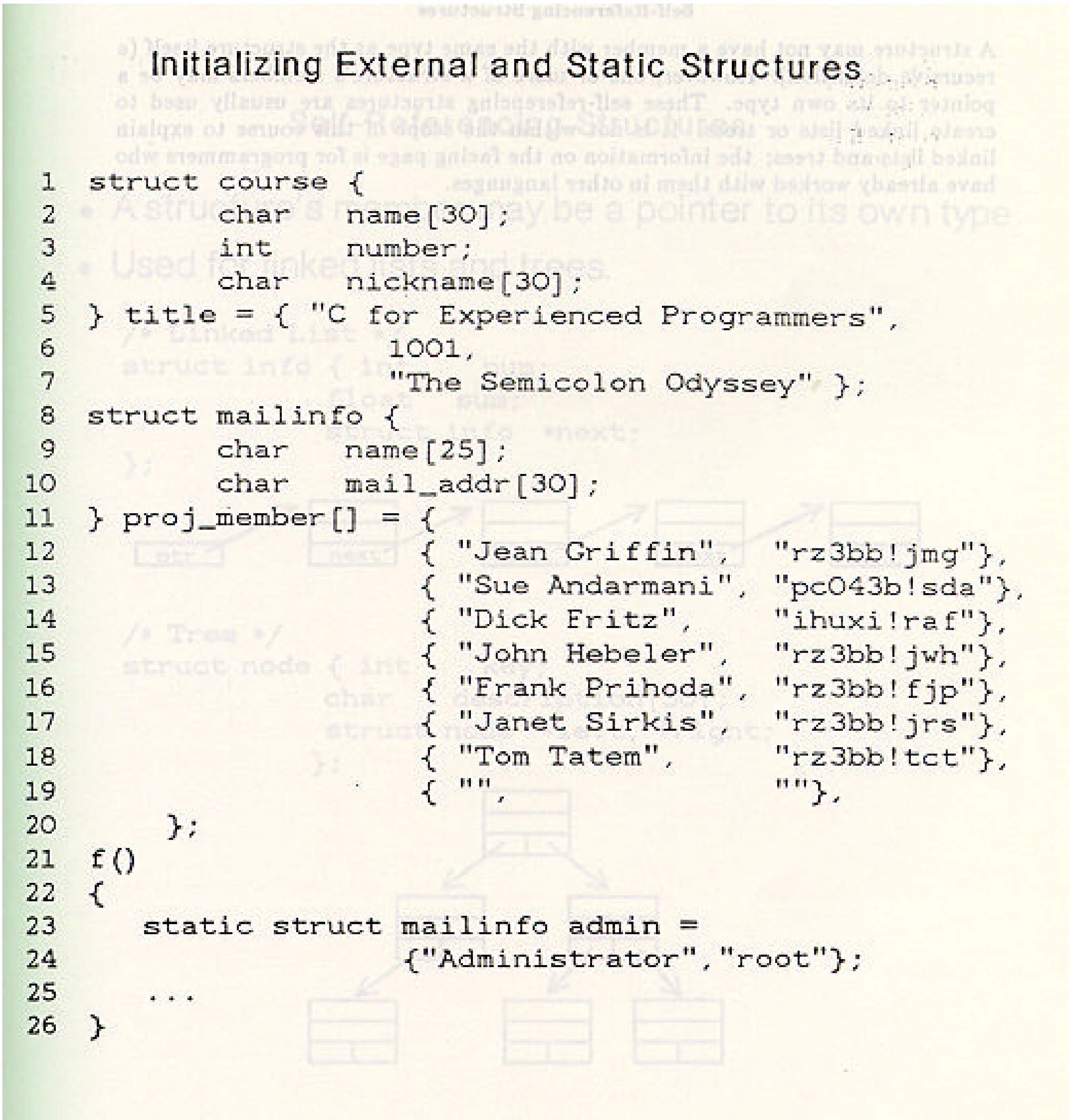
Exercise – Pointers to Structures

The previous program would run more efficiently if a structure pointer was passed to raise(). How would the program change if a pointer was used? Replace the ________s below with the correct statements.

```
#include  <stdio.h>
#include  “emp.h”
main()
{
  struct  emp  prgmr ;
    _____  raise() ;
  printf(“Old salary: $%.0f\n”, prgmr.salary) ;
    ______  raise( ______ prgmr,  .12) ;
  printf(“New salary: $%.0f\n”, prgmr.salary) ;
 }

      ______   raise(person, increase)
 struct  emp  person ;
 double  increase ;
 {
     person________salary  *=  (1 + increase) ;
     ________
 }
```

```
Output :
Old salary : $3100
New salary : $3472
```



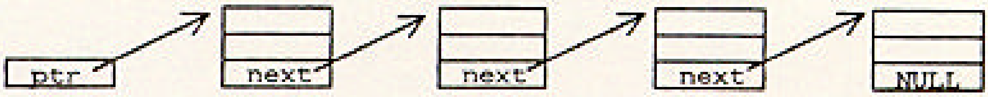
Initializing External and Static Structures

```
struct course {
       char   name[30];
       int    number;
       char   nickname[30];
} title = { "C for Experienced Programmers",
                1001,
                "The Semicolon Odyssey" };
struct mailinfo {
       char   name[25];
       char   mail_addr[30];
} proj_member[] = {
                   { "Jean Griffin",     "rz3bb!jmg"},
                   { "Sue Andarmani",    "pc043b!sda"},
                   { "Dick Fritz",       "ihuxi!raf"},
                   { "John Hebeler",     "rz3bb!jwh"},
                   { "Frank Prihoda",    "rz3bb!fjp"},
                   { "Janet Sirkis",     "rz3bb!jrs"},
                   { "Tom Tatem",        "rz3bb!tct"},
                   { "",                 ""},
    };
f()
{
   static struct mailinfo admin =
                   {"Administrator","root"};
   ...
}
```

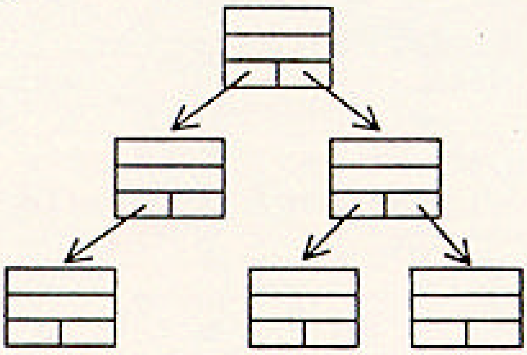
Self-Referencing Structures

- A structure's member may be a pointer to its own type
- Used for linked lists and trees.

```
/* Linked List */
struct info { int      num;
              float   sum;
              struct info  *next;
};
```



```
/* Tree */
struct node { int     key;
              char    description[50];
              struct node  *left, *right;
             };
```

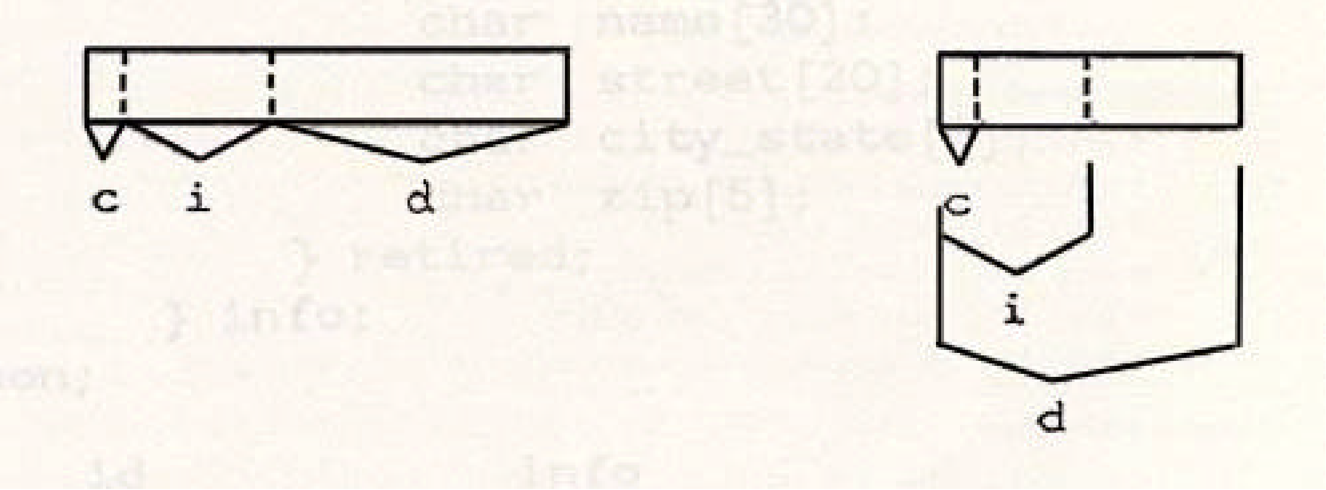


Unions

Union vs. Structures

- Declaration, tag, template same as structures except keyword union used.
- Can hold only one member at a time.
- Big enough to hold largest member.

```
struct  s_tag  {                          union  u_tag  {
                char      c ;                             char      c ;
                int       i ;                             int       i ;
                double    d ;                             double    d ;
             }  s_item ;                               }  u_item ;
```



```
printf("The size of s_item is %d\n", sizeof(s_item)) ;
printf("The size of u_item is %d\n", sizeof(u_item)) ;
```

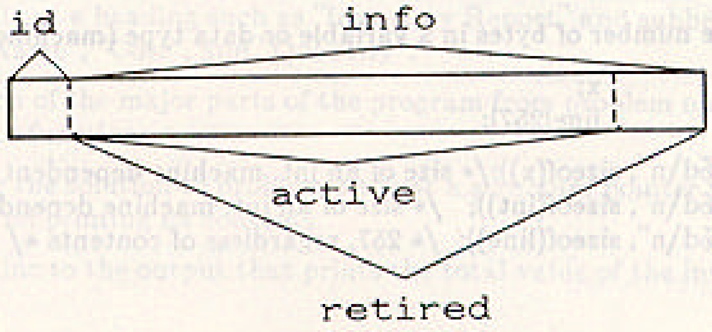
```
Output :
The size of s_item is 16
The size of u_item is 8
```

Advantages of Unions

- Flexibility - May hold objects of different types
- Often used to save space

```
/* Holds info about active/retired employees */

struct mail {
          char id;   /* a-active, r-retired */
          union  {
               struct {
                   char  name[30];
                   char  dept[10];
                   char  location[3];
               } active;
               struct {
                   char  name[30];
                   char  street[20];
                   char  city_state[3];
                   char  zip[5];
               } retired;
          } info;
} person;
```


id
info
active
retired