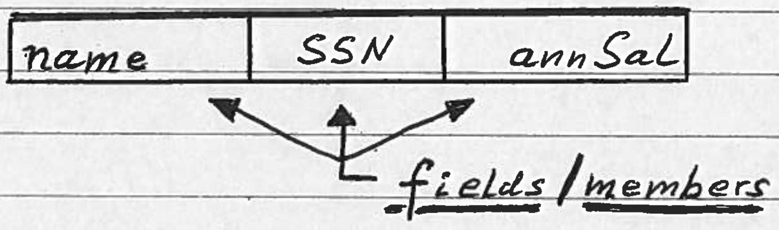
# RECORDS

• **Ex:**

| Instructor | Quarter | Course |
|---|---|---|
| "Bernd Hamann" | "SQ 14" | "ECS 30-A" |

"compound data structure"

"Employee":



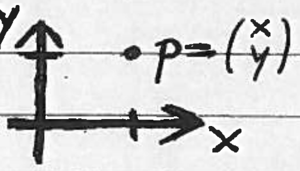
```
➡ typedef struct { string name;
                   string SSN;
                   double annSal;
                 }
          employeeT;          /* definition */

➡ employeeT  emp;             /* declaration */

➡ emp.name   = "Bill Chen";
  emp.SSN    = "111-22-3333";
  emp.annSal = 75000.0;       /* selection */

OR: employeeT emp

      = {"Bill Chen", "111-22-3333", 75000.0};
```

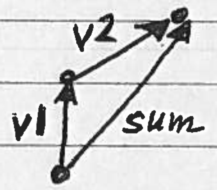
• **Ex:** 

```
typedef struct { double x;
                 double y;
               }
               point2dT;
```

➡ Function can return point2dT:

```
point2dT DefinePoint (double x, double y)
{
   point2dT  p;
   p.x  = x;
   p.y  = y;
   return(p);
}
```

➡ Records as arguments of a function:



```
point2dT VectorSum ( point2dT v1,
                     point2dT v2 )
{
   point2dT   sum;

   sum.x  = v1.x + v2.x;
   sum.y  = v1.y + v2.y;
   return (sum);
}
```

## Records and Arrays

• **Ex:**

| | name | SSN | annSal |
|---|---|---|---|
| emp[0] | | | |
| emp[1] | | | |
| ... | | | |
| emp[9] | | | |

```
employeeT  emp[10];
                /*declaration*/

emp[3].name = "U2";
                /*selection*/
```

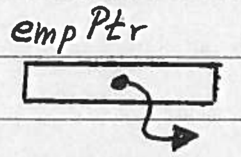
→ "Database" of employees: (array-based)

```
... employeeT  emp[MaxEmps];
    int        noEmps;
    ...
```

```
/* List all employees */
void ListEmps( employeeT  emp[],
               int        noEmps )
{ int  i;

  for ( i=0 ; i<noEmps ; i++)
  {
     printf("%s (%s) %lf \n",
              emp[i].name, emp[i].SSN,
                    emp[i].annSal );
  }
}
```

## Records and Pointers

→ 

pointer to record allocated after 'New' command

ⓘ

```
typedef struct {...}
        employeeT;

employeeT  *empPtr;
```

**OR:**

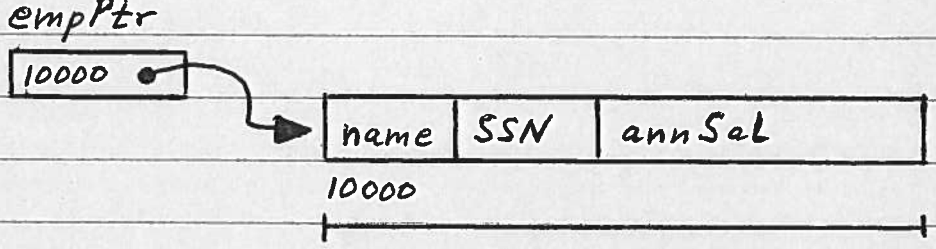
ⓘⓘ

```
typedef struct {...}
        *employeeT;

employeeT  empPtr;
              /* is a pointer by definition*/
```

→ Memory allocation for records:

```
employeeT  empPtr;      /* pointer-based definition ⓘⓘ */
...
empPtr = New (employeeT);  /*argument of 'New': */
                           /* must be a pointer! */
                           /* 'New' returns pointer */
                           /* to Large-enough memory. */
```



size of an employee record

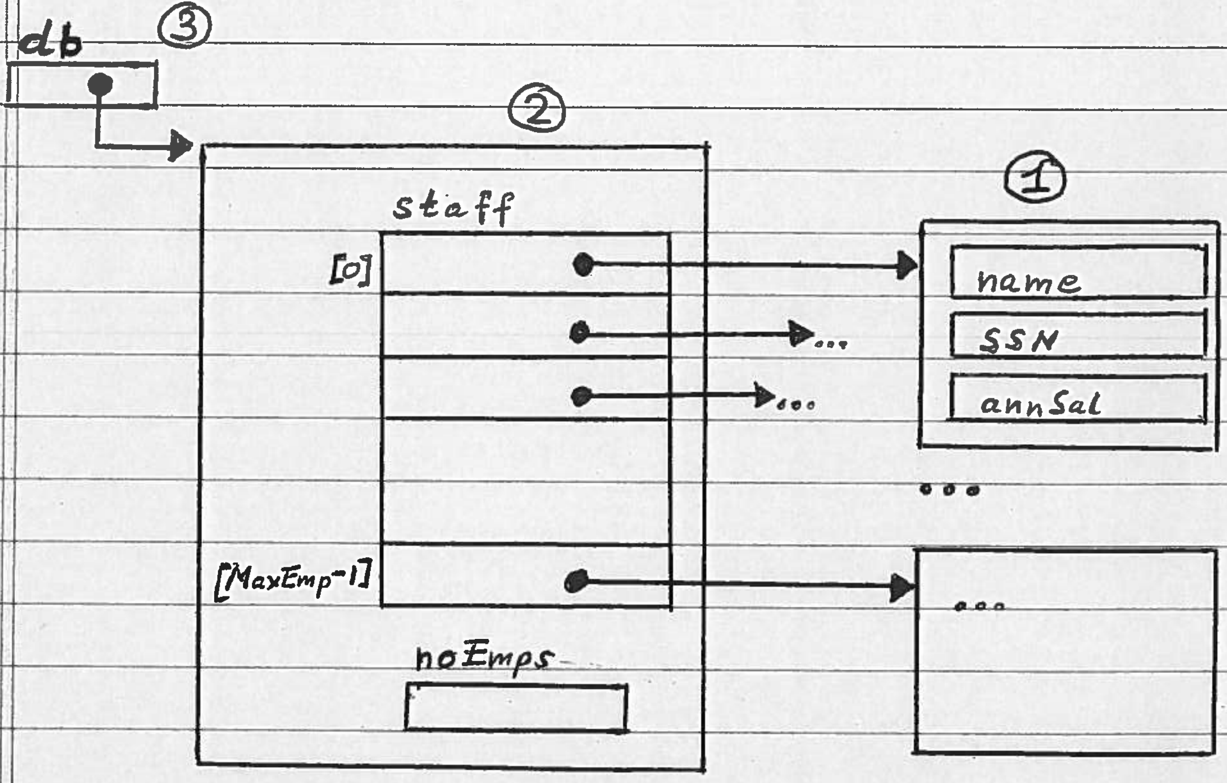
→ De-referencing and selection combined:

! • Ex: `empPtr -> annSal      /* refers to an employee's */`
`                              /* annual salary           */`

```
empPtr -> annSal = 75000.0;
    EQUIVALENT TO:
(*empPtr).annSal = 75000.0;
```

## An Employee Database – Top-down Design of Functionality

→ Conceptual design:



"Data structure design of database"

→ design algorithms for this data structure!

➡ Definitions and declarations:

① 
```
typedef struct { string name;
                 string SSN;
                 double annSal;
               }
          * employeeT ;
```

② 
```
typedef struct { employeeT staff[MaxEmp];
                 int       noEmps;
               }
          * empDB ;
```

③ 
```
empDB db ;
```

➡ Algorithms for: Ⓐ Reading, Ⓑ Printing, Ⓒ Manipulating

Ⓑ 
```
/* Printing DB */
void ListEmps ( empDB db ) /* input: pointer to DB */
{ int i;

  for (i=0; i < db->noEmps; i++)
  { printf ("%s (%s) %lf \n",
                 db -> staff [i] -> name,
                 db -> staff [i] -> SSN,
                 db -> staff [i] -> annSal );
  }
}
```

↑ pointer pointing to a pointer pointing to name/SSN/annSal of i-th employee

Ⓐ /*Reading DB */

```
emp DB ReadEmps (void)          /*output: pointer to DB */
{
    emp DB     db;                         /* pointer to DB */
    employeeT  emp;                        /* pointer to emp record */
    int        noEmps;

    db = New (empDB);   /* memory alloc. for array of pointers *
                          /* to employee records and noEmps field *

    noEmps = 0;
    printf ("Enter employee data - DONE: NO INPUT. \n");
    while ((emp = ReadOneEmp ()) != NULL)
    {
        db -> staff [noEmps] = emp;
                                  /* assigning pointer variable */

        noEmps ++;
    }
    db -> noEmps = noEmps;
    return (db);
}
```

➡ define 'ReadOneEmp' function!

```
/* Read data for one employee */
employeeT ReadOneEmp (void)
                                        /* output: pointer to */
{                                       /* one new emp record */
  employeeT    emp ;
  string       name;

   printf ("Name: ");
   name =  GetLine();
   if ( StringLength (name)==0)   /*end of input */
        return (NULL);

   emp = New ( employeeT);   /* allocate mem. for */
                              /* data for one employee */
    emp -> name = name;
    printf ("SSN: ");
    emp -> SSN = GetLine();
    printf ("Annual salary: ");
    emp -> annSal = GetReal();

    return(emp);
}
```

```
©/*Manipulating DB: raising salary by 5% */
 void AdjustSalary ( empDB db)
 { int i;
   for ( i=0; i< db->noEmps; i++)
   { db -> staff[i]->annSal *= 1.05;
 }  }
```

• **Ex: An interactive math program**

➡ Functionality:

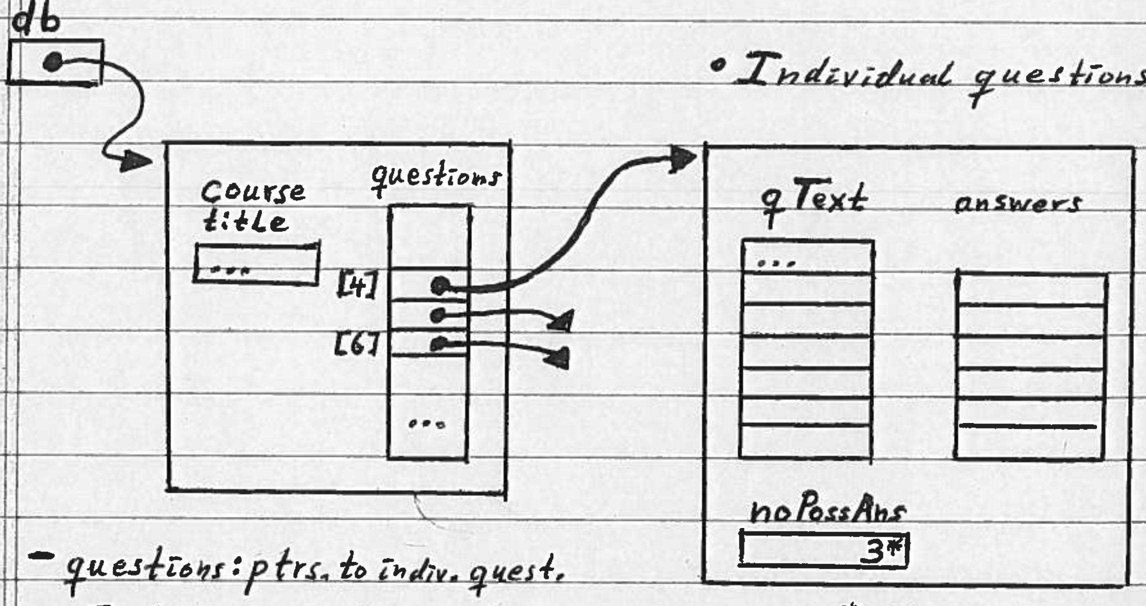
1) Ask a question, e.g., "What is 14*28?"
2) Obtain answer.
3) Depending on answer, ask next question.

➡ Algorithms to be developed:

1) Select question and display as string.
2) Read answer as string.
3) Determine correctness of answer — and determine type of next question.

➡ Design of data structures:

• Individual questions



- questions: ptrs. to indiv. quest.
- qText: array of strings (= Lines of one question)
- noPossAns: no. of possible answers
- answers: array of "answer type" – e.g.:

| ansText | nextQuestion |
|---|---|
| string | int |

*effective size of answers-array is 3.

➡ Possible file data structure/format:

```
1
True or false? The Earth is flat.
...
 true: 2        /*go to quest. 2*/
 false: 3       /*go to quest. 3*/

2
 ...
```

Fig. 16-1