Structures

What is it ?

- A collection of information about something
- Example : Information about students
  - Name
  - Roll no
  - Hostel address
  - Home address
  - Contact number
- Each element in the collection can be of different datatype

Syntax

```
struct student {
  char name[15];
  char roll_num[10];
  float cpi;
}sleeping, awake;
```

- Student is a tag and not a datatype
- Sleeping and awake are structure variables
- Another method:
  - Typedef struct { ....} student;
- Accessing values:
  - Sleeping.name, awake.cpi

Structure within structure

```
struct date {
  int day;
  int month;
  int year;
};
struct person {
  char name[20];
  struct date dob;
  float salary;
};
```

Struct date can be accessed by other parts of program

```
struct person {
  char name[20];
  struct date {
    int day;
    int month;
    int year;
  } dob;
  float salary;
};
```

Struct date cannot be accessed elsewhere

Structure within structure

```
struct date {
  int day;
  int month;
  int year;
};
struct person {
  char name[20];
  struct date dob;
  float salary;
};
```

```
struct person {
  char name[20];
  struct date {
    int day;
    int month;
    int year;
  } dob;
  float salary;
};
```

```
struct person p;

printf(“%d”, p.dob.year);
printf(“%s”, p.name);
```

Array of structures

```
struct date {
  int day;
  int month;
  int year;
};
struct person {
  char name[20];
  struct date dob;
  float salary;
};
```

```
struct person empRec[10];

// Accessing structure
// elements
// within the array

printf(“%s”,
  empRec[4].name);
printf(“%d”,
  empRec[5].dob.year);
```

Array within structure

```
struct customer {
  char name[20];
  int age;
  float salary;
  int acc_nos[10];
} custrec;

// custrec has information of a customer who has
// multiple accounts with an organisation

printf(“the account info in 3rd location is: %d”,
       custrec.acc_nos[3]);
```

Creating user degined datatypes: by using `typedef`

- Syntax:

`typedef ExistingTypeName NewTypeName;`

- `typedef int length;`
  - `length` is now synonym for int
  - `length` is now a type name and not a variable
  - `length l1, l2;`
- `typedef int emprec[10];`
  - `emprec p1, p2;`
  - `emprec` is new data type which is a 10 element integer array
  - p1 and p2 are variables of emprec datatype and have 10 element integer arrays. Accessed as: p1[4], &p1[0], ...

```
typedef struct studstruct
{
  int rollno;
  int subject;
  int marks;
} student;

struct studstruct s1;
 -- same as --
student s1;

simple method:
typedef struct {
...
} student;
```

```
typedef struct {
  int day; int month;
  int year; } date;

struct person{
  char name[30];
  date birthday;
  float salary;
} emprec[10];
    --or --
typedef struct pers{
  char name[30];
  date birthday;
  float salary;
} person;
person emprec[10]; //usage
```

Pointers to structures

```
struct date{...};

struct person
{
  char name[30];
  struct date dob;
  float salary;
} emprec;

// usage
struct person p1, *ptr;
```

```
struct person *ptr;

ptr = &emprec;

// accessing members
ptr-> salary
ptr-> name
ptr->name[5]
ptr->dob.day
```

Pointers within structures

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

void main()
{
  float x;

  typedef struct
  {
    int day;
    int month;
    int year;
  } date;
```

```
typedef struct
{
  char name[20];
  char *lastname;
  date birthday;
  float *salary;
} person;

person emprec;

person *ptr = &emprec;
date *birth =
&emprec.birthday;
```

```
strcpy(emprec.name, "Harry");
emprec.lastname = "Potter";

ptr->birthday.day=24;
emprec.birthday.month = 7;
birth->year = 90;

x=7000.0;
ptr->salary = &x;
printf("Employee Details:\n");
printf("\nName: %s %s",
emprec.name, ptr->lastname);

printf("Birthday: %3d %3d %3d",
emprec.birthday.day,
(*ptr).birthday.month,
birth->year);
```

```
// parenthesis: are required
// above for *ptr, as * has
// lower precedence than . (dot)
// If we omit parenthesis it
// will have effect of:
//  *(ptr.birthday.month)
// which is meaningless
printf("\nSalary: %6.2f\n",
*emprec.salary);
}
```

```
strcpy(emprec.name, "Harry");
emprec.lastname = "Potter";

ptr->birthday.day=24;
emprec.birthday.month = 7;
birth->year = 90;

x=7000.0;
ptr->salary = &x;
printf("Employee Details:\n");
printf("\nName: %s %s",
emprec.name, ptr->lastname);

printf("Birthday: %3d %3d %3d",
emprec.birthday.day,
(*ptr).birthday.month,
birth->year);
```

```
// parenthesis: are required
// above for *ptr, as * has
// lower precedence than . (dot)
// If we omit parenthesis it
// will have effect of:
//   *(ptr.birthday.month)
// which is meaningless
printf("\nSalary: %6.2f\n",
*emprec.salary);
}
```

```
Employee Details:

Name: Harry Potter
Birthday:  24   7  90
Salary: 7000.00
```

Pass structure to function

- Define structures for person
- Inside main():
  - Declare: person emprec
  - Initialise emprec
  - Print values in emprec
  - Call function readInput(emprec) to read new values
  - Call function printOutput(emprec) to print the values
    - These should be values from the readInput function

Return structure from function

- Define structure person
- In main()
  - Define person p1;
  - Call function to read data: p1 = readIn();
  - Call function to print values: printOutput(p1);
- In readIn()
  - Define person readIn() {emprec record; ...}
  - Read user inputs and assign to structure variable record
  - Return structure variable record to main function

Pass array of structure to function

```
#include <stdio.h>

typedef struct {
  int day;
  int month;
  int year;
} date;

typedef struct {
  char name[20];
  date birthday;
  int  salary;
} person;
```

```
int highest(person rec[],
int);
void printOut(person);

void main() {
  int j;
  person records[] =
    {
      {"AAA",25,1,80,1000},
      {"BBB",17,2,90,5000},
      {"CCC",10,3,98,2000}
    };

  j = highest(records, 3);
printf("\nThe highest salary
   being paid is to index:
   %4d\n", j);

printOut(records[j]); }

void printOut(person p) {
  printf("\nEmployee
   Details:\n");
  printf("\nName: %s,
    Birthday: %3d %3d %3d,
    Salary:%4d\n",
    p.name, p.birthday.day,
    p.birthday.month,
    p.birthday.year,
    p.salary); }
```

```
int highest(person q[], int m)
{
  int i, j;
  int max;

  max = q[0].salary;

  for(i=1 ; i< m ;i++)
  {
    if(max < q[i].salary)
    {
      max = q[i].salary;
      j = i; }
  }
  return j;
}
```

```
printf("\nThe highest salary
 being paid is to index: %4d\n", j);

printOut(records[j]); }

void printOut(person p) {
  printf("\nEmployee Details:\n");
  printf("\nName: %s,
    Birthday: %3d %3d %3d,
    Salary:%4d\n",
    p.name, p.birthday.day,
    p.birthday.month,
    p.birthday.year,
    p.salary); }
```

```
int highest(person q[], int m)
{
  int i, j;
  int max;

  max = q[0].salary;

  for(i=1 ; i< m ;i++)  {
    if(max < q[i].salary) {
      max = q[i].salary;
      j = i; } }
  return j;
}
```

```
The highest salary being paid is to index:   1

Employee Details:

Name: BBB, Birthday:  17   2  90, Salary:5000
```

Return pointers to structures

```
#include <stdio.h>

typedef struct {
  int day;
  int month;
  int year;
} date;

typedef struct {
  char name[20];
  date birthday;
  int  salary;
} person;
```

```
person* highest(person rec[],
int);
void printOut(person*);

void main() {
  int j;
  person *ptrPerson;
  person records[] =
    {
      {"AAA",25,1,80,1000},
      {"BBB",17,2,90,5000},
      {"CCC",10,3,98,2000}
    };
ptrPerson = highest(records, 3);
```

```
printf("\nThe highest salary
being paid is: %4d\n",
ptrPerson->salary);

printOut(ptrPerson); }

void printOut(person *p) {
  printf("\nEmployee
   Details:\n");
  printf("\nName: %s,
    Birthday: %3d %3d %3d,
    Salary:%4d\n",
    p->name, p->birthday.day,
    p->birthday.month,
    p->birthday.year,
    p->salary); }
```

```
person* highest
(person q[], int m)
{
  int i, j;
  int max;

  max = q[0].salary;

  for(i=1 ; i< m ;i++)
  {
    if(max < q[i].salary)
    {
      max = q[i].salary;
      j = i; }
  }
  return (&q[j]);
}
```

```
printf("\nThe highest salary
being paid is: %4d\n",
ptrPerson->salary);

printOut(ptrPerson); }

void printOut(person *p) {
  printf("\nEmployee
   Details:\n");
  printf("\nName: %s,
    Birthday: %3d %3d %3d,
    Salary:%4d\n",
    p->name, p->birthday.day,
    p->birthday.month,
    p->birthday.year,
    p->salary); }
```

```
person* highest
(person q[], int m)
{
  int i, j;
  int max;

  max = q[0].salary;

  for(i=1 ; i< m ;i++)
  {
    if(max < q[i].salary)
    {
      max = q[i].salary;
      j = i; }
  }
  return (&q[j]);
}
```

```
The highest salary being paid is :   5000

Employee Details:

Name: BBB, Birthday:  17   2  90, Salary:5000
```

Unions

- Normally variables are stored in memory and occupy different locations/addresses
- Often there are variables which are not required at same time. Ex. Variable filename is used for some time and later output string variable is used and filename is not used. But memory is allocated to both. To optimise, we can use same storage space to keep these variables one at a time
- Overlay of one or more variables in the same memory area is called a union
- Compiler allocates sufficient storage to accommodate the largest element. Other elements use the same space
- Writing one will overwrite the other

Syntax same as structures

```
union desc {
  char name[20];
  float salary;
  int idno;
};

union desc var1, *var2;
var2 = &var1;
// usage
var1.name;
var1.idno;
var2->salary;
```

Operations on elements

- ALL structure elements can be accessed at any point of time
- Only ONE member of a union may be accessed at any given time. Other variables will have garbage
- The valid variable is the most recently written
- Programmer has to keep track of active variable

Scope of element in union

```
#include <stdio.h>

struct stest {
  int   i;
  char  c;
  float f;
};

union test {
  int   i;
  char  c;
  float f;
};
```

```
void main() {
  union test u;
  struct stest s;

  printf("\nsize of struct
   stest is: %d\n", sizeof(s));
  printf("\nsize of union test
    is: %d\n", sizeof(u));

  u.i = 10;
  u.c = 'A';
// c is active, it will overwrite i
  u.f = 5.5;
}
```

Scope of element in union

```
struct stest {
  int   i;
  char  c;
  float f; };

union test {
  int   i;
  char  c;
  float f; };

Main() { ...
  u.i = 10;
  u.c = 'A';
  u.f = 5.5;
}
```

```
size of struct stest is: 12

size of union test is: 4

After u.i=10, value of i is: 10

After u.c='A', value of c is: A

After u.c='A', value of i is: 65

After u.f=5.5, value of f is: 5.500000

After u.f=5.5, value of c is:

After u.f=5.5, value of i is: 1085276160
```