typedef and structures

typedef

אוניברסיטת אריאל בשומרון

- typedef is often used in the declaration of a structure type.
- typedef is used to define new types
- The benefit - more readable no need to use the word **struct** when declaring variables.
- General syntax –

```
typedef <existing_type_name> <new_type_name>;
```

- typedef usually comes before the main function
- For example -
  - typedef int distance ;
  - typedef double weight ;
- From that point, declaring a variable as a weight will be equivalent to declaration of the variable as double. Same as with int distance

Example


```
#include <stdio.h>
typedef  double  distance ;
void main(void)
{
    distance  miles , kmeters ;
    scanf("%lf", &miles);
    kmeters = miles * 1.609 ;
    printf("%.2f miles = %.2f kmeters\n", miles , kmeters );
}
```

Output

2.00 miles = 3.22 kilometers

Example

- Define 2 new types – vector and matrix
- vector is a double 1-dimensional array and matrix is a double 2-dimensional array.
- Define 2 new vectors and 1 new matrix of these types

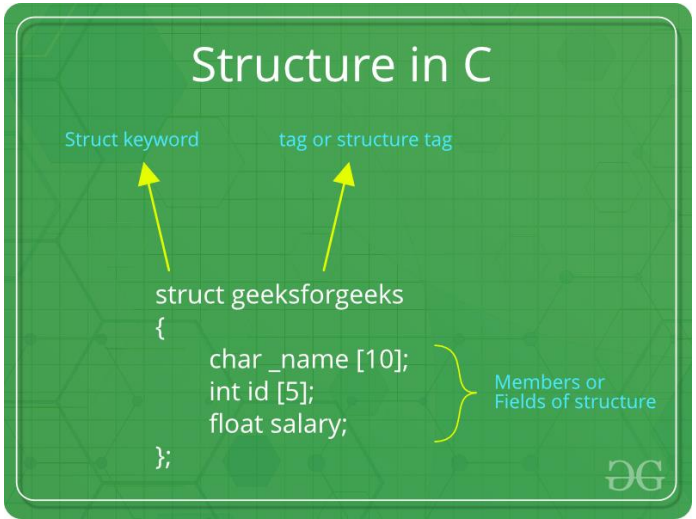
```
#define  N   10
#define  M   20
typedef  double   vector[M] ;
typedef  double   matrix[N][M] ;

vector   a , b ;   /* a & b are both arrays dimension M */
matrix   mat ;    /* mat is a two-dimensions array N×M */
```

אוניברסיטת
אריאל
בשומרון

Structures

Structures

- A struct is a collection of variables, gathered into one super-variable
- These variables' types can be the same (as a normal array) or different from each other (the key role of structures)
- Structure members may be ordinary variable types, but also other structures and even arrays !
- It is used to define more complex data types
- Variables in a struct are called members or fields



Structure declaration

- To define a structure, you must use the **struct** statement.
- The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows –

```
struct [structure tag] {

   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

- **Structure tag** is optional and each member definition is a simple variable such as int i; or float f; or array, another struct or any other variable.
- At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional.

```
struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
} book;
```

Struct declaration


```
struct person
{
    int age;
    char name[30];
};
```

Tag name

type

struct person

```
struct
{
    int age;
    char  name[30];
} per,  *p_per;
```

variables

No tag name.

per and **p_per** (a pointer !) are declared upon the creation of the new type.

Struct declaration

```
struct person
{
   int age;
   char name[30];
} per;
```

Tag Name

variable

type variable

```
struct person per;
```

```
typedef struct person
{
   int age;
   char name[30];
 } per;
```

Tag Name

synonym

type variable

```
person   per;
```

This declaration is similar to :

```
struct person per;
```

Example – homogeneous structure

אוניברסיטת אריאל בשומרון

- The following example shows homogeneous data struct

```
struct complex {
        int real;
        int imaginary;
        };
```

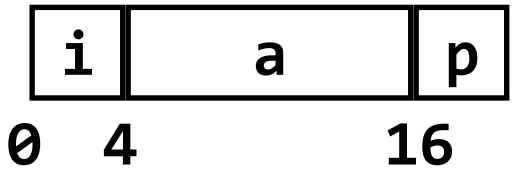
- Once we define a structure, we can treat it as a new type.
  - In a program, we can then write:

```
struct complex num1, num2, num3;
```

Heterogeneous structure

- Structure members are of different types
  - Example:

```
struct rec
{
   int i;
   int a[3];
   int *p;
};
```



Struct initialization

- Structure members cannot be initialized with declaration.
- For example the following C program fails in compilation.

```
struct Point
{
    int x = 0;  // COMPILER ERROR:  cannot initialize members here
    int y = 0;  // COMPILER ERROR:  cannot initialize members here
};
```

- The reason for above error is simple, when a datatype is declared, no memory is allocated for it. Memory is allocated only when variables are created.
- Structure members **can be** initialized using curly braces '{}'. For example, following is a valid initialization.

Struct initialization

- Struct can be initialized in a way similar to arrays

```
struct  person {
        char  name[N] , address[M];
        long       id ;
        int  age , status ;
} ;
```

```
struct rec
{
  int i;
  int a[3];
  int *p;
};
```

```
struct person class[2] = {{“Yaakov Cohen”,”Lahavim”, 012000678, 56, 1},
                          {“Cohen OneTwo”,”Meitar”, 012333678, 44, 2}};
```

```
int k;
struct rec r = { 5, { 0,1,2}, &k };
```

Struct initialization

אוניברסיטת אריאל בשומרון

```c
#include<stdio.h>

struct Point
{
    int x, y, z;
};

int main()
{
    // Examples of initialization using designated initialization
    struct Point p1 = {.y = 0, .z = 1, .x = 2};
    struct Point p2 = {.x = 20};

    printf ("x = %d, y = %d, z = %d\n", p1.x, p1.y, p1.z);
    printf ("x = %d", p2.x);
    return 0;
}
```

Accessing structure members

אוניברסיטת אריאל בשומרון

- To access any member of a structure, we use the **member access operator (.)**.
- The member access operator is coded as a period between the structure variable name and the structure member that we wish to access.
- The keyword **struct** defines variables of structure type.

Example


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

struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

int main( ) {

   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info */
   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);

   /* print Book2 info */
   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);

   return 0;
}
```

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

Structure as function argument

- We can pass a structure as a function argument in the same way as we pass any other variable or pointer.

```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

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

struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

/* function declaration */
void printBook( struct Books book );

int main( ) {

   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info */
   printBook( Book1 );

   /* Print Book2 info */
   printBook( Book2 );

   return 0;
}

void printBook( struct Books book ) {

   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);
}
```

Typedef & Struct

- typedef is often used in the declaration of a structure type.
- Example:

```
typedef struct emp {
        char name [25];
        double salary;
} employee, *pEmployee;
```

- struct emp is the type and employee is the variable – now they are the same.
- struct emp*, employee* and pEmployee are the same.

Typedef & struct – 3 options

אוניברסיטת אריאל בשומרון

```
struct  person {
    char  name[N] , address[M];
    long   id ;
    int  age , status ;
} guy1, guy2;
```

Step 1 – person type definition

```
typedef struct  person{
    char  name[N] , address[M];
    long          id ;
    int  age , status ;
} guy1, guy2;
```

Step 2 – person variable definition

```
struct  person{
    char  name[N] , address[M];
    long   id ;
    int  age , status ;
} ;

typedef  struct  person guy1;
```

Pointers to structures

אוניברסיטת אריאל בשומרון

- Many times we prefer to pass structures to functions by address, and not by value. Thus a new copy of the structure is not created – just a pointer to the existing structure
- We can define pointers to structures in the same way as we define pointer to any other variable

```
struct Books *struct_pointer;
```

- We can store the address of a structure variable in the above defined pointer variable.
- To find the address of a structure variable, place the '&'; operator before the structure's name as follows −

```
struct_pointer = &Book1;
```

Pointers to structures

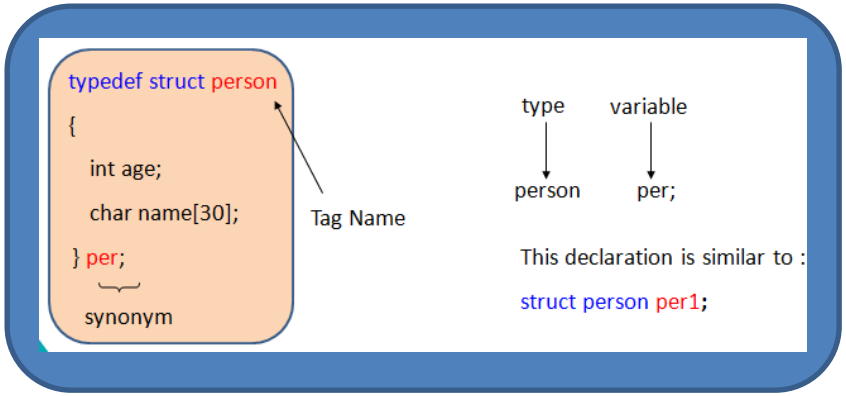
- To access the members of a structure using a pointer to that structure, we have two options (pp1 is the pointer to per) –
  - must use the '→' operator as follows –

    pp1-> name

    pp1 -> age
  - Reference a field via pp1 (the pointer to per1)

    (*pp1).name

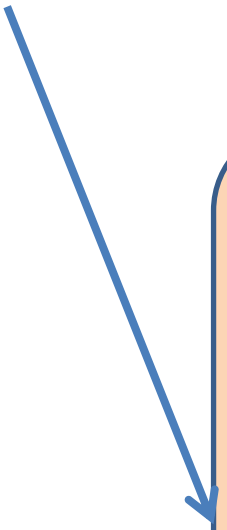
    (*pp1).age



Access to struct members via pointers

אוניברסיטת אריאל בשומרון

```
typedef struct _MyStr
{
   int _a[10];
} MyStr;
```



```
void main()
{
   MyStr x;
   MyStr *p_x = &x;
   x._a[2] = 3;
   (*p_x)._a[3] = 5;
   p_x->_a[4] = 6;
}
```

Rewrite the books example using pointers


```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

Rewrite the example using pointers


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

struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

/* function declaration */
void printBook( struct Books *book );
int main( ) {

   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info by passing address of Book1 */
   printBook( &Book1 );

   /* print Book2 info by passing address of Book2 */
   printBook( &Book2 );

   return 0;
}

void printBook( struct Books *book ) {

   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);
}
```

```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

Example

- Write a program that gets a struct called *person* and a pointer to the struct, asking the user to enter age and weight and insert these details into the relevant struct fields (by using the pointer)

Example

- Write a program that gets a struct called *person* and a pointer to the struct, asking the user to enter age and weight and insert these details into the relevant struct fields (by using the pointer)

```c
#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;
}
```

Exercise

אוניברסיטת אריאל בשומרון

- Enter the desired number of persons
- For each person get “name” and “age” and print them all – using pointers

Solution

```c
#include <stdio.h>
#include <stdlib.h>
struct person {
   int age;
   float weight;
   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);
```

```c
   printf("Displaying Information:\n");
   for(i = 0; i < n; ++i)
       printf("Name: %s\tAge: %d\n", (ptr+i)->name, (ptr+i)->age);

   return 0;
}
```

Malloc - sometimes, the number of struct variables you declared may be insufficient. You may need to allocate memory during run-time. Here's how you can achieve this in C programming.

Solution


```
typedef struct  complex{
    double x,y;
}complex;

complex  * mult_comp(complex a,  complex b){
    complex  * z =  (complex *)malloc(sizeof(complex));
    z->x = a.x * b.x – a.y * b.y;
    z->y = a.x * b.y + a.y * b.x;
    return z;
}
```

Nested structures

אוניברסיטת אריאל בשומרון

```
typedef  struct {
    char    name[30];
    char    address[50];
    char    phone_num[15];
} Supplier ;
```

Step 1 – Supplier type definition

Step 2 – Inventory type definition

```
typedef struct {
    int max;
    int min;
    int curr;
} Inventory;
```

Step 3 – Item type definition that uses Supplier and Inventory

```
typedef struct {
    char    name[30];
    Supplier   supplier;
    char    depart [15];
    Inventory   inventory;
 } Item ;
```

Nested Structures

- A member of a structure may itself be a structure.
- For example, a worker is naturally a person, and so is a student.

Step 1 – person type definition

```
typedef struct {
char f_name[15];
char l_name[15];
long id_num;
} person;
```

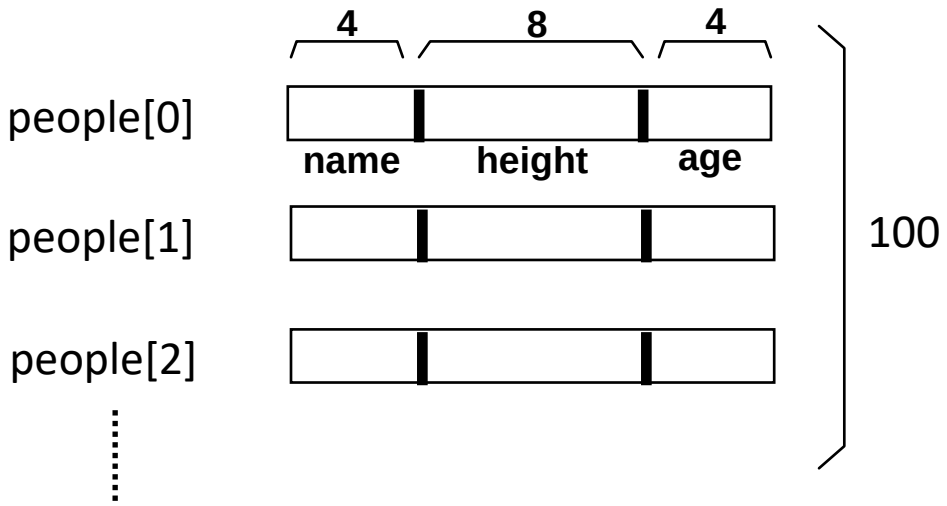
Step 2 – using person

```
typedef  struct{
  person per;
  int experience_years;
} worker;
```

```
typedef struct{
  person per;
  int grade;
} student;
```

Array of structs

```
struct person{
        char *name;
        double height;
        int age;
} people[100];
struct person  *p_person;
p_person = people;
```



- There are many ways to access the data member age in an element :
  - people[2].age
    - p_person[2].age
      - (people + 2)->age
        - (p_person +2)->age
          - (*(people + 2)).age
            - (*(p_person + 2)).age

Array of structs

אוניברסיטת אריאל בשומרון

```c
#include<stdio.h>

struct Point
{
    int x, y;
};

int main()
{
    // Create an array of structures
    struct Point arr[10];

    // Access array members
    arr[0].x = 10;
    arr[0].y = 20;

    printf("%d %d", arr[0].x, arr[0].y);
    return 0;
}
```

Structs limitation

- We cannot use operators like +,- etc. on Structure variables.
- For example, consider the following code:

```
struct number
{
    float x;
};
int main()
{
    struct number n1,n2,n3;
    n1.x=4;
    n2.x=3;
    n3=n1+n2;
    return 0;
}

/*Output:

prog.c: In function 'main':
prog.c:10:7: error:
invalid operands to binary + (have 'struct number' and 'struct
  n3=n1+n2;

*/
```

Comparison and Copying

אוניברסיטת אריאל בשומרון

- Comparison - structures cannot be compared using the == operator
  - They must be compared member by member
  - Usually this will be done in a separate function
- Copying - structures can be copied using the '=' operator
  - Member-wise copy

Comparison and Copying

Two variables of the same structure type can be copied the same way as ordinary variables. If persona1 and person2 belong to the same structure, then the following statements Are valid.

person1 = person2;

person2 = person1;

C does not permit any logical operators on structure variables In case, we need to compare them, we may do so by comparing members individually.

person1 = = person2

person1 ! = person2

Statements are not permitted.

Comparison and Copying


```
structclass
   {
        int  number;
        char name[20];
        float marks;
   };

   main()
   {
        int  x;
        structclass student1 = {111,"Rao",72.50};
        structclass student2 = {222,"Reddy", 67.00};
        structclass student3;

        student3 = student2;

        x = ((student3.number ==  student2.number) &&
             (student3.marks  ==  student2.marks)) ? 1 : 0;

        if(x == 1)
       {
             printf("\nstudent2 and student3 are same\n\n");
            printf("%d %s %f\n", student3.number,
                                 student3.name,
                                 student3.marks);
        }
        else
            printf("\nstudent2 and student3 are different\n\n");

   }
```

Output

```
   student2 and student3 are same

   222 Reddy 67.000000
```

Structures containing arrays

אוניברסיטת אריאל בשומרון

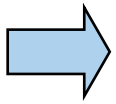
- A structure member that is an array does not 'behave' exactly like an ordinary array.
- When copying a structure that contains an array member, the array is copied element by element
  - Not just the address gets copied.
  - Reminder – ordinary arrays can't be copied simply by using the '=' operator. They must be copied using a loop.

Arrays in structs copying

אוניברסיטת אריאל בשומרון

- Copy struct using '=', just struct values

```
Vec a,b;
a._arr[0] = 5;
a._arr[1] = 3;
b = a;
```



a

_arr = {5,3,?,…}

b

_arr = {5,3,?,…}

Structures and arrays as arguments

אוניברסיטת אריאל בשומרון

- Passing a structure to a function – by a value
- When an array is passed as an argument to a function, the address of the 1st element is passed.
- Structs are passed by value, exactly as the basic types.

Pointers in structs copying

- The result:

```
Vec a,b;
a._arr[0] = 5;
a._arr[1] = 3;
a._p_arr = a._arr;
b = a;
*(b._p_arr) = 8;

printf ("%f", a._arr[0]);
```

```
// output
8
```