Structures

Palash Dey
Department of Computer Science & Engg.
Indian Institute of Technology
Kharagpur

Slides credit: Prof. Indranil Sen Gupta

What is a Structure?

- It is a convenient tool for handling a group of logically related data items.
  - Examples:
    - Student name, roll number, and marks.
    - Real part and complex part of a complex number.
- This is our first look at a non-trivial data structure.
  - Helps in organizing complex data in more meaningful way.
- The individual elements of a structure are called *members*.

Defining a Structure

- A structure may be defined as:

```
struct tag {
             member 1;
             member 2;
             :
             member m;
          };
```

  - *struct* is the required keyword.
  - *tag* is the name of the structure.
  - *member 1*, *member 2*, … are individual member declarations.

Contd.

- The individual members can be ordinary variables, pointers, arrays, or other structures.
  - The member names within a particular structure must be distinct from one another.
  - A member name can be the same as the name of a variable defined outside of the structure.
- Once a structure has been defined, the individual structure-type variables can be declared as:

```
  struct tag var_1, var_2, …, var_n;
```

Example

- A structure definition:

```
struct student {
                  char   name[30];
                  int   roll_number;
                  int   total_marks;
                  char   dob[10];
               };
```

- Defining structure variables:

```
struct student  a1, a2, a3;
```

A new data-type

A Compact Form

- It is possible to combine the declaration of the structure with that of the structure variables:

```
struct tag {
              member 1;
              member 2;
              :
              member m;
           }  var_1, var_2,…, var_n;
```

- In this form, ***tag*** is optional.

Equivalent Declarations

```
struct student  {
                   char  name[30];
                   int  roll_number;
                   int  total_marks;
                   char  dob[10];
                }  a1, a2, a3;
```

```
struct          {
                   char  name[30];
                   int  roll_number;
                   int  total_marks;
                   char  dob[10];
                 } a1, a2, a3;
```

Processing a Structure

- The members of a structure are processed individually, as separate entities.
- A structure member can be accessed as:

```
variable.member
```

where `variable` refers to the name of a structure-type variable, and `member` refers to the name of a member within the structure.

- Examples:

```
a1.name, a2.name, a1.roll_number,
a3.dob
```

Example: Complex number addition

```
#include  <stdio.h>
main()
{
       struct  complex
       {
          float  real;
          float  cmplex;
       }  a, b, c;

       scanf ("%f %f", &a.real, &a.cmplex);
       scanf ("%f %f", &b.real, &b.cmplex);

       c.real = a.real + b.real;
       c.cmplex = a.cmplex + b.cmplex;
       printf ("\n %f + %f j", c.real, c.cmplex);
}
```

Comparison of Structure Variables

- Unlike arrays, group operations can be performed with structure variables.
  - A structure variable can be directly assigned to another structure variable of the same type.

    ```
    a1 = a2;
    ```

    - All the individual members get assigned.
  - Two structure variables can be compared for equality or inequality.

    ```
    if (a1 == a2)……
    ```

    - Compare all members and return 1 if they are equal; 0 otherwise.

Arrays of Structures

- Once a structure has been defined, we can declare an array of structures.

```
struct student class[50];
```

  - The individual members can be accessed as:

```
class[i].name
class[5].roll_number
```

Arrays within Structures

- A structure member can be an array:

```
struct  student
{
       char  name[30];
       int  roll_number;
       int  marks[5];
       char  dob[10];
}  a1, a2, a3;
```

- The array element within the structure can be accessed as:

  `a1.marks[2]`

Defining data type: using *typedef*

- One may define a structure data-type with a single name.
- General syntax:

```
typedef struct {
                       member-variable1;
                       member-variable2;
                       .
                       member-variableN;
                    } tag;
```

- *tag* is the name of the new data-type.

typedef : An example

```
typedef struct {
                  float real;
                  float imag;
               }  _COMPLEX;

_COMPLEX a, b, c;
_COMPLEX complexarray[100];
```

A new data type

Structure Initialization

- Structure variables may be initialized following similar rules of an array. The values are provided within the second braces separated by commas.
- An example:

```
_COMPLEX a={1.0,2.0}, b={-3.0,4.0};
```

⇩

```
a.real=1.0;    a.imag=2.0;
b.real=-3.0;   b.imag=4.0;
```

Parameter Passing in a Function

- Structure variables can be passed as parameters like any other variables. Only the values will be copied during function invocation.

```
void swap (_COMPLEX a, _COMPLEX b)
 {
    _COMPLEX tmp;

    tmp = a;
    a = b;
    b = tmp;
 }
```

An Example

```
#include <stdio.h>

typedef struct {
                    float real;
                    float imag;
         } _COMPLEX;

void swap (_COMPLEX a, _COMPLEX b)
 {
    _COMPLEX tmp;

    tmp = a;
    a = b;
    b = tmp;
 }
```

Example:: contd.

```
void print (_COMPLEX a)
 {
       printf("(%f, %f) \n“, a.real, a.imag);
 }

main()
 {
       _COMPLEX x = {4.0,5.0}, y = {10.0,15.0};

       print(x); print(y);
       swap(x,y);
       print(x); print(y);
 }
```

- **Output:**

```
      (4.000000, 5.000000)
      (10.000000, 15.000000)
      (4.000000, 5.000000)
      (10.000000, 15.000000)
```

  - No swapping takes place, since only values are passed to the function. The original variables in the calling function remains unchanged.

Returning structures

- It is also possible to return structure values from a function. The return data type of the function should be same as the data type of the structure itself.

```
_COMPLEX add (_COMPLEX a, _COMPLEX b)
 {
    _COMPLEX tmp;

    tmp.real = a.real + b.real;
    tmp.imag = a.imag + b.imag;

    return(tmp);
 }
```

Direct arithmetic operations are not possible with structure variables.

Example: Addition of two complex numbers

```c
#include <stdio.h>

typedef struct {
                      float real;
                      float imag;
         } _COMPLEX;


_COMPLEX add (_COMPLEX a, _COMPLEX b)
 {
    _COMPLEX tmp;

    tmp.real = a.real + b.real;
    tmp.imag = a.imag + b.imag;

    return(tmp);
 }
```

```c
int main()
{
  _COMPLEX num1, num2, sum;
  scanf ("%f %f", &num1.real,
                      &num1.imag);
  scanf ("%f %f", num2.real,
                      &num2.imag);

  sum = add (num1, num2);
  printf ("\nSum is: %f + j %f",
    sum.real, sum.imag);

}
```

Example: Compute perimeter of polygon

```
#include <stdio.h>

typedef struct {
                 int sides;
                 float length[10];
         } POLYGON;


float perimeter (POLYGON p)
 {
    float peri = 0.0;
    int i;

    for (i=0; i<p.sides; i++)
      peri += p.length[i];

    return(peri);
 }
```

```
int main()
{
  POLYGON shape;
  int k;
  float peri;

  scanf (”%d”, &shape.sides);
  for (k=0; k<shape.sides; k++)
    scanf (”%f", &shape.length[k];

  peri = perimeter (shape);
  printf ("\nPerimeter is: %f",
                           peri);
}
```

Estimating the Size of a Structure

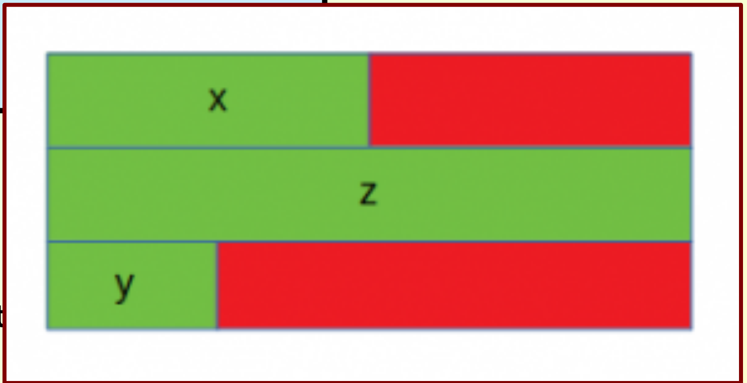
- The "sizeof" for a struct variable is not always equal to the sum of the "sizeof" of each individual member.
  - Padding is added by the compiler to avoid alignment issues.
  - Padding is only added when a structure member is followed by a member with a larger size or at the end of the structure.
- Exact convention may vary from one compiler to another.

(a) Example 1

```
#include <stdio.h>
int main()
{
    struct A {
            // sizeof(int) = 4
        int x;
            // Padding of 4 bytes
            // sizeof(double) = 8
        double z;
            // sizeof(short int) = 2
        short int y;
            // Padding of 6 bytes
    };
    printf("Size of struct: %ld", sizeof(struct A));
    return 0;
}
```

Here, x (int) is followed by z (double), which is larger in size than x. Hence padding is required after x. Also, padding is required at the end for data alignment.

```
Size of struct: 24
```

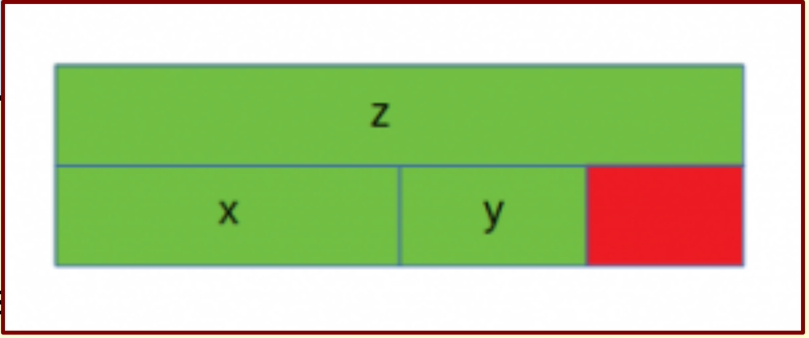


(b) Example 2

```
#include <stdio.h>
int main()
{
    struct B {
            // sizeof(double) = 8
        double z;
            // sizeof(int) = 4
        int x;
            // sizeof(short int) = 2
        short int y;
            // Padding of 2 bytes
    };
    printf("Size of struct: %ld", sizeof(struct B));
    return 0;
}
```

The members of the structure are sorted in decreasing order of their sizes. Hence padding is required only at the end.

```
Size of struct: 16
```

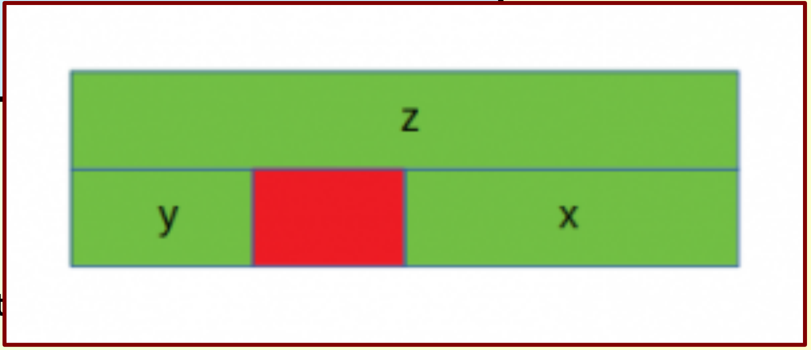


(c) Example 3

```
#include <stdio.h>
int main()
{
    struct C {
            // sizeof(double) = 8
        double z;
            // sizeof(short int) = 2
        short int y;
            // Padding of 2 bytes
            // sizeof(int) = 4
        int x;
    };
    printf("Size of struct: %ld", sizeof(struct B));
    return 0;
}
```

Here, y (short int) is followed by x (int) and hence padding is required after y. No padding is required at the end.

```
Size of struct: 16
```



Exercise Problems

1. Extend the complex number program to include functions for addition, subtraction, multiplication, and division.
2. Define a structure for representing a point in two-dimensional Cartesian co-ordinate system.
    - Write a function to compute the distance between two given points.
    - Write a function to compute the middle point of the line segment joining two given points.
    - Write a function to compute the area of a triangle, given the co-ordinates of its three vertices.
3. Define a structure to represent students' information (name, roll number, cgpa). Read the data corresponding to N students in a structure array, and find out the students with the highest and lowest cgpa values.