Programming and Algorithms

COMP1038.PGA

Session 13 & 14: Structures

Outline

- Introduction
- Structure Definitions
- Initializing Structures
- Accessing Members of Structures
- Using Structures with Functions
- Typedef
- Union

Introduction

- Variables of primitive data types can only hold one value of a particular data type
- Array can hold multiple values of same data type
- In real life, data of different types is organized together
- Such as, Address book , library book information, student information, etc.
- Structure and Union are solution for this.

Structures

- Collections of related variables (aggregates) under one name
- Can contain variables of different data types
- Commonly used to define records to be stored in files
- Combined with pointers, can create linked lists, stacks, queues, and trees

Syntax Structure definition

```
struct <structure name>
{
structure element 1 ;
structure element 2 ;
structure element 3 ;
……
} ;
```

Structure Definitions

- Example

```
struct Student {
        char name [20];
        char address [20];
        int rollno;
        char class[5];
        char year[20];
 };
```

- struct introduces the definition for structure Student
- Student is the structure name and is used to declare variables of the structure type

Structure Definitions cont…

- struct information
  - A struct cannot contain an instance of itself
  - Can contain a member that is a pointer to the same structure type
  - A structure definition does not reserve space in memory
    - Instead creates a new data type used to declare structure variables
- Declarations
  - Declared like other variables:

```
Student oneStud, Stud[66], *SPtr;
```

- Can use a comma separated list:

```
struct Student {
                char name [20];
                char address [20];
    }oneStud, Stud[66], *SPtr;
```

Declaration of Struct variable

-
  ```
  struct book
  {
    char name ;
    float price ;
    int pages ;
  } b1, b2, b3 ;
  ```

-
  ```
  Struct
  {
   char name ;
   float price ;
   int pages ;
  } b1, b2, b3 ;
  ```

```
struct book
{
char name ;
float price ;
int pages ;
} ;
struct book b1, b2, b3
```

Structure Definitions cont…

- Valid Operations
  - Assigning a structure to a structure of the same type
  - Taking the address (&) of a structure
  - Accessing the members of a structure
  - Using the *sizeof* operator to determine the size of a structure

Initializing Structures

- Initialize lists
  - Example:

```
Student S1 = { “Ashish", “Pune‟ 2543, “SEV”, ”2020-2021 ”};
```

```
struct book
{
  char name[10] ;
  float price ;
  int pages ;
} ;
struct book b1 = { "Basic", 200, 550 } ;
struct book b2 = { "Physics", 300, 800 } ;
```

Assignment statements

- Example:

```
Student S2 = S1;
```

- Could also declare and initialize as follows:

```
Student S2;
S2.name = “Ashish”;
S2.address = “Pune”;
S2.rollno = 2543;
S2.class = “SEV”;
s2.year=“2020-2021 ”;
```

Accessing Members of Structures

- Accessing structure members
  - Dot operator (.) used with structure variables

```
Student S1;
printf( "%s", S1.name );
```

  - Arrow operator (->) used with pointers to structure variables

```
Student *SPtr = &S1;
printf( "%s", SPtr->name );
```

  - SPtr->name is equivalent to

```
( *SPtr ).name;
```

Memory allocation

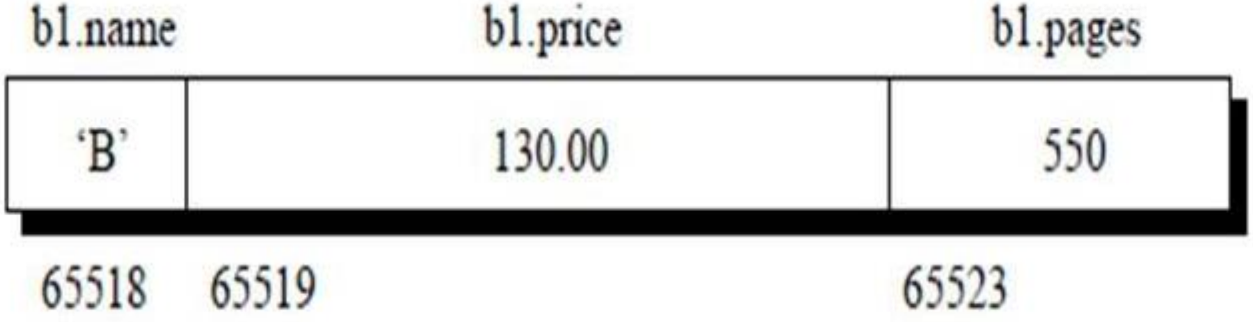
```
/* Memory map of structure
elements */
main( ) {
struct book
{
  char name ;
  float price ;
  int pages ;
} ;
struct book b1 = { 'B’,130.00 , 550 } ;
```

```
printf ( "\nAddress of name =
%u", &b1.name );
printf ( "\nAddress of price =
%u", &b1.price ) ;
printf ( "\nAddress of pages =
%u", &b1.pages ) ; }
```

Here is the output of the program…

```
Address of name = 65518
Address of price = 65519
Address of pages =65523
```

Memory allocation cont…



b1.name b1.price b1.pages

'B' 130.00 550

65518 65519 65523

Array of Structures

```
/* Usage of an array of
structures */
main( )
{
   struct book
   {
      char name ;
      float price ;
      int pages ;
   }
} ;
struct book b[100] ;
int i ;
```

```
for ( i = 0 ; i <= 99 ; i++ )
{
   printf ( "\nEnter name, price and
               pages " ) ;
   scanf ( "%c %f %d", &b[i].name,
              &b[i].price, &b[i].pages ) ;
}
printf ( "\n%c %f %d", b[i].name,
b[i].price, b[i].pages ) ;
```

Additional Features of Structures

1. Structure and Assignment operator

- The values of a structure variable can be assigned to another structure variable of the same type using the assignment operator.
- No piecemeal copy is needed .

```
struct employee
{
    char name[10] ;
    int age ;
    float salary ;
} ;
```

Additional Features of Structures cont…

Structure and Assignment operator

```
struct employee e1 = { "Sanjay", 30, 10000.00} ;
struct employee e2, e3 ;
/* piece-meal copying */
strcpy ( e2.name, e1.name ) ;
e2.age = e1.age ;
e2.salary = e1.salary ;
/* copying all elements at one go */
e3 = e2 ;
printf ( "\n%s %d %f", e1.name, e1.age, e1.salary ) ;
printf ( "\n%s %d %f", e2.name, e2.age, e2.salary ) ;
printf ( "\n%s %d %f", e3.name, e3.age, e3.salary ) ;
```

Additional Features of Structures cont…

2. Nested structure

One structure can be nested within another structure. Using this facility complex data types can be created. The following program shows nested structures at work.

```
    void main( )
    {
        struct address
        {
           char phone[15];
           char city[25];
           int pin;
        };
struct emp
{
  char name[25];
  struct address a;
};
struct emp e = { "jeru",
"531046", "nagpur", 10 };
 printf ( "\nname = %s phone
 = %s", e.name, e.a.phone );
 printf ( "\ncity = %s pin =
 %d", e.a.city, e.a.pin );
}
```

Additional Features of Structures cont…

3. Structures With Functions

- Passing structures to functions
  - Pass entire structure
  - Or, pass individual members
  - Both pass call by value
- To pass structures call-by-reference
  - Pass its address
  - Pass reference to it

Additional Features of Structures cont…

3. Structures With Functions Call by values

- Like an ordinary variable, a structure variable can also be passed to a function.
- Or whole structure can be passed to a function

Additional Features of Structures cont…

3. Passing individual structure elements

```
 main( )
 {
  struct book
  {
     char name[25] ;
    char author[25] ;
    int callno ;
  } ;
 struct book b1 = { "Let us C", "YPK", 101 } ;
display ( b1.name, b1.author, b1.callno ) ;
}
```

```
void display ( char *s, char *t, int n )
 {
    printf ( "\n %s %s %d", s, t, n ) ;
}
```

And here is the output…

```
Let us C YPK 101
```

Additional Features of Structures cont…

3. Passing entire structure variable at a time

```
struct book
{
   char name[25] ;
   char author[25] ;
   int callno ;
} ;
main()
{
  struct book b1 = { "Let us C", "YPK", 101 } ;
  display ( b1 ) ;
}
```

```
display ( struct book b )
{
printf ( "\n%s %s %d", b.name,
b.author, b.callno ) ;
}
```

output…

```
Let us C YPK 101
```

Structure and pointer

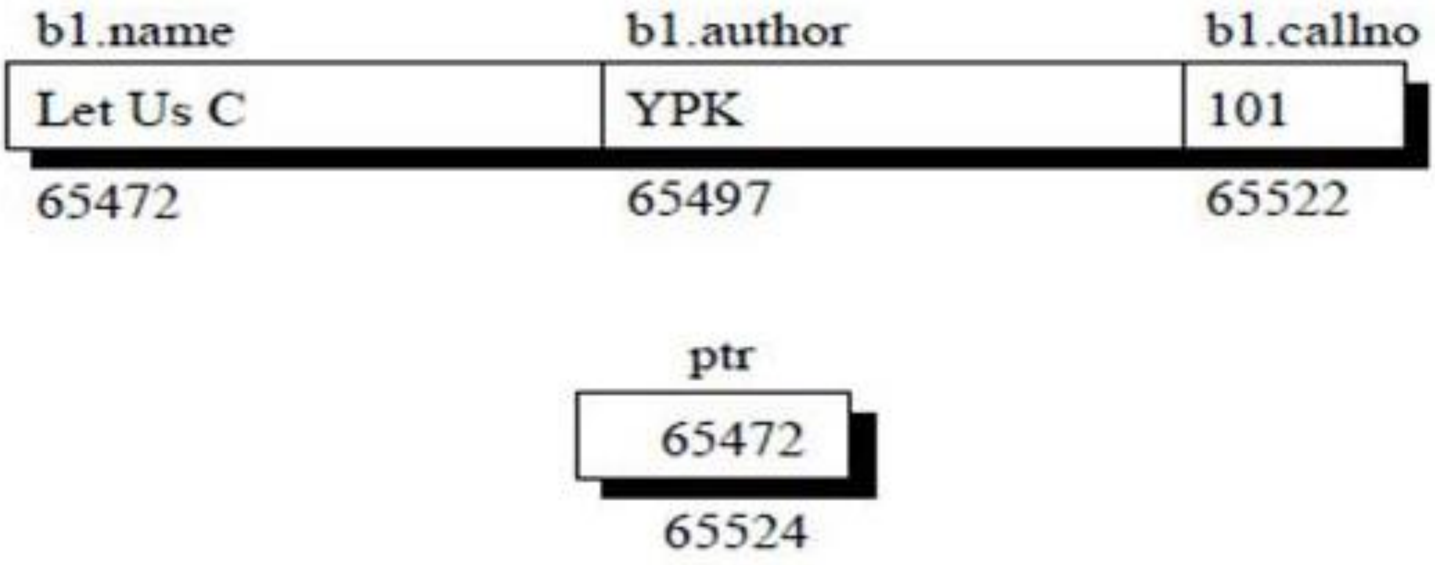
- The way we can have a pointer pointing to an int, or a pointer pointing to a char.
- similarly we can have a pointer pointing to a struct.
- Such pointers are known as ‘structure pointers’

Structure and pointer cont…

```
Void main ()
{
struct book
char name[25] ;
char author[25] ;
int callno ;
} ;
```

```
struct book b1 = { "Let us C",
"YPK", 101 } ;
struct book *ptr ;
ptr = &b1 ;
printf ( "\n%s %s %d", b1.name,
b1.author, b1.callno ) ;
printf ( "\n%s %s %d", ptr->name,
ptr->author, ptr->callno ) ;
```

Structure and pointer memory map



b1.name b1.author b1.callno

Let Us C YPK 101

65472 65497 65522

ptr

65472

65524

Call by reference and structure

```
/* Passing address of a
structure variable */

struct book
{
char name[25] ;
char author[25] ;
int callno ;
} ;
```

```
void main()
{struct book b1 = { "Let us C",
"YPK", 101 } ;
display ( &b1 );
 }
display ( struct book *b )
{ printf ( "\n%s %s %d", b->
name, b->author, b->callno ) ;
}
```

And here is the output…

```
Let us C YPK 101
```

Array of structure and pointer

```
struct book
{ char name[25] ;
char author[25] ;
int callno ;
} ;

void main()
{ int n;
struct book b1[20] ;
n= input (b1);
display ( b1,n );
}
```

```
void display(struct book *b , int n) {
int l;
printf(“\n\t\t Books Available “ );
printf(“|n\t Sr. No \t Name \t Aruthor
\t Call No \n\n“ );
for(i=0;i<n;i++)
{ pritnf(“\n\t\t %d \t %s \t%s \t %d“
            ,i+1, *(b+i)->name, *(b+i)->
          author,*(b+i)->callno);
}
```

Input function

```
int input(struct book *b )
{int I , n;
printf(“\n\t\t Enter No of books “);
scanf(“%d”,&n);
for(i=0;i<n;i++)
{  printf(“|n\t Sr. No %d “, i+1);
   printf(“\n\t Name :::”);
   flushall();
   gets(*(b+i) -> name);
   printf(“\n\t Author Name :::”);
   flushall();
   gets(*(b+i) -> author);
   printf(“\n\t Call No \n\n“ );
   scanf(“ %d “,&(b+i)->callno);
}
return n;
}
```

Uses of Structures

- Data base management
- Changing the size of the cursor
- Clearing the contents of the screen
- Placing the cursor at an appropriate position on screen
- Drawing any graphics shape on the screen
- Receiving a key from the keyboard
- Checking the memory size of the computer
- Finding out the list of equipment attached to the computer
- Formatting a floppy
- Hiding a file from the directory
- Displaying the directory of a disk
- Sending the output to printer
- Interacting with the mouse

Thank you!

What would be the output of the following program

```
void main( )
{
 struct gospel
 { int num ;
   char mess1[50];
   char mess2[50];
 } m ;
 m.num = 1 ;
 strcpy ( m.mess1, "If all that you have is hammer" );
 strcpy ( m.mess2, "Everything looks like a nail" );
 /* assume that the strucure is located at address 1004 */
 printf ( "\n%u %u %u", &m.num, m.mess1, m.mess2 ) ;
}
```

What would be the output of the following program

```
struct gospel {
int num ;
char mess1[50] ;
char mess2[50] ;
} m1 = { 2, "If you are driven by success", "make sure that it is a quality
drive“ } ;
Void main( )
{
  struct gospel m2, m3 ;
  m2 = m1 ;
  m3 = m2 ;
  printf ( "\n%d %s %s", m1.num, m2.mess1, m3.mess2 ) ;
}
```

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Point out the errors, if any, in the following programs:

```
void main( )
{ struct employee
  { char name[25];
    int age;
    float bs;
  };
  struct employee e;
  strcpy ( e.name, "Hacker" );
  age = 25;
  printf ( "\n%s %d", e.name, age );
}
```

Point out the errors, if any, in the following program:

```
void main( )
{ struct
  { char name[25];
    char language[10];
  };
  struct employee e = { "Hacker", "C" };
  printf ( "\n%s %d", e.name, e.language );
}
```

Point out the errors, if any, in the following program:

```
struct virus
{ char signature[25];
  char status[20];
  int size;
} v[2] = { "Yankee Doodle", "Deadly", 1813, "Dark Avenger",
"Killer", 1795 };

void main( )
{ int i;
  for ( i = 0 ; i <=1 ; i++ )
     printf ( "\n%s %s", v.signature, v.status );
}
```

typedef

- typedef
  - Creates aliases for previously defined data types
  - Use typedef to create shorter type names
- Typedef allows us to associate a name with a structure (or other data type).

typedef

- Put typedef at the start of your program.

```
typedef struct line {
  int x1, y1;
  int x2, y2;
} LINE;

void main()
{
  LINE line1;
}
```

line1 is now a structure of line type

Typedef Example

```
typedef struct
{ char *first;
  char *last;
  char SSN[9];
  float gpa;
  char **classes;
} student;
student stud_a, stud[20] ,*sptr;
```

Typedef Example

```
struct employee
{ char name[25];
  int age;
  float bs;
}
typedef empoyee emp;
```

Union

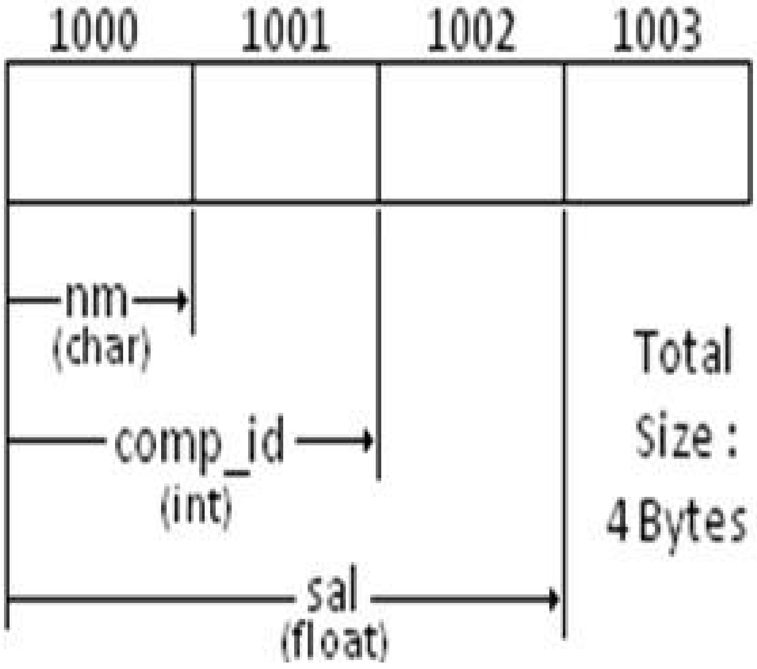
- Like structures, but every member occupies the same region of memory!
  - Structures: members are “and”ed together: “name and species and owner”
  - Unions: members are “xor”ed together

```
union VALUE
{ float f;
  int i;
  char *s;
};
```

Union

- Storage
  - size of union is the size of its largest member
  - avoid unions with widely varying member sizes;
  - for the larger data types, consider using pointers instead
- Initialization
  - Union may only be initialized to a value appropriate for the type of its first member

Memory allocation



```
union techno
{ int comp_id;
  char nm;
  float sal;
}tch;
```

Accessing element:
tch.comp_id
tch.nm
tch.sal

Union usage

- variable format input records (coded records)
- sharing an area to save storage usage
- unions not used nearly as much as structures

Thank you!