Structures

ESC108A Elements of Computer Science and Engineering
B. Tech. 2017

Course Leaders:

Roopa G.

Ami Rai E.

Chaitra S.

Objectives

- At the end of this lecture, student will be able to
  – Use ***typedef*** and ***struct*** keywords in C programming language
  – Apply pointers to structures
  – Explain self-referential structures and their applications

Contents

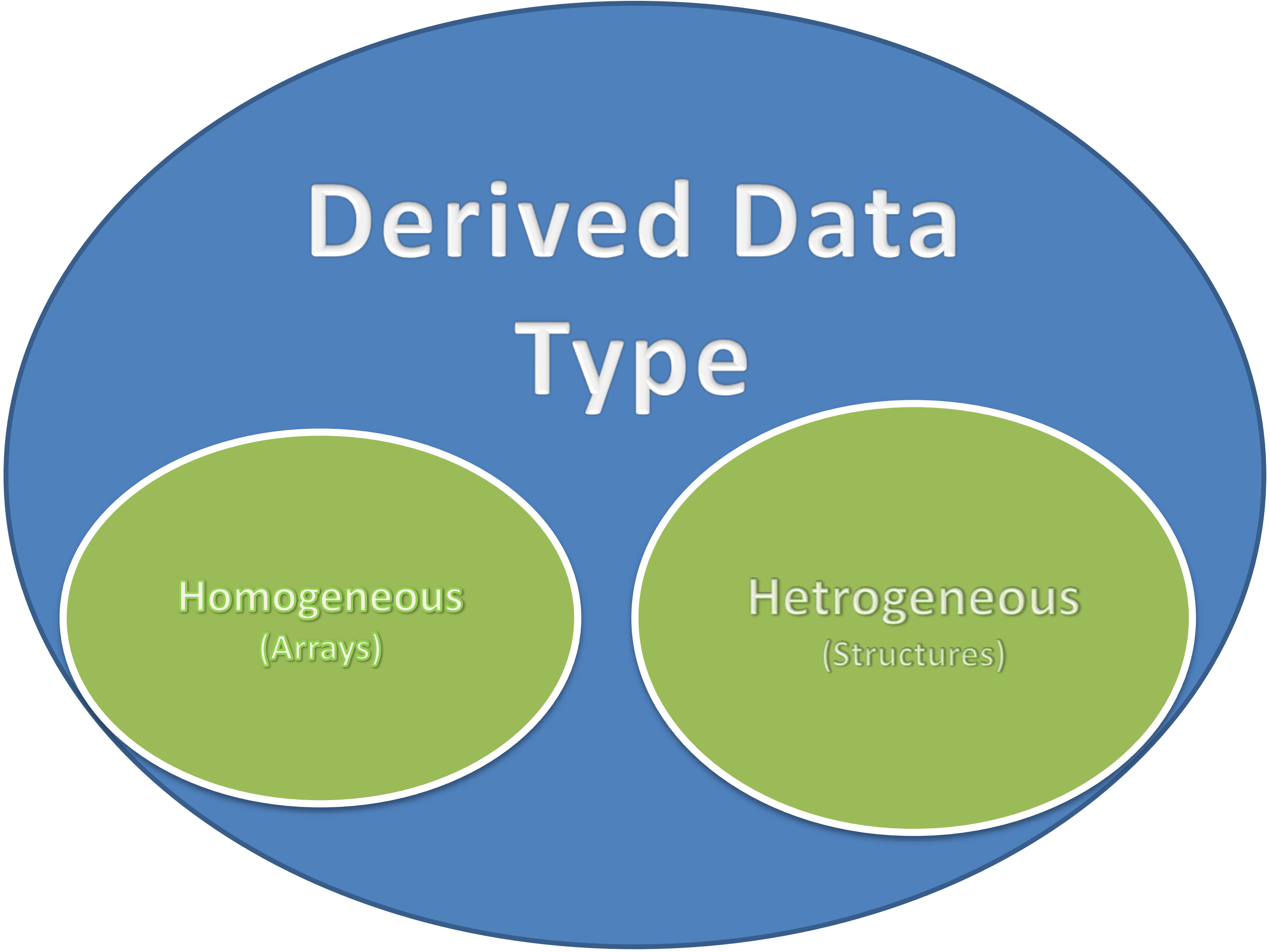
- Structure definition
- Structure variable declarations
- Pointers to structures
- Union

Question

- All students have same characteristics
  - Name
  - Roll number
  - Age
  - …

- Why is there no mechanism to relate them in a programming language?

Derived Data Type



Structures

- Collections of related variables 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 data structures such as linked lists, stacks, queues, and trees

Structure Definition

- Keyword struct introduces the structure definition
- Structure definition does not reserve space in memory
  - Creates a new data type that is used to declare structure variables

Structure Definition - Example

```
struct student{
        char name[10];
        char rollNum[10];
         int age;
 };
```

- student is the structure name and is used to declare variables of the structure type
- student contains three members
  - two members of type char - name and rollNum
  - a member of type int - age

Structure Definition – Example contd.

- A structure cannot contain an instance of itself
- Can contain a member that is a pointer to the same structure type

```
struct employee2 {
        char firstName[20];
        int age;
        char gender;
        double hourlySalary;
        struct employee2 person; /* ERROR */
        struct employee2 *ePtr; /* pointer */
}
```

Structure Variable Declaration

- Declared like other variables

```
struct student myStudent;
struct student students[60]; //Array of structures
```

- Variables can be declared as part of the structure definition

```
struct student{
    char name[10];
    char rollNum[10];
   int age;
 } myStudent, students[60];
```

Initializing Structures

- Initializer lists

```
STUDENT myStudent = {“sarma”, “CJB0912332”, 31};
```

- Assignment statements

```
STUDENT sarma = myStudent;
```

Or

```
STUDENT sarma;
sarma.name = “Sarma”;
sarma.rollNo= “CJB0912332”;
sarma.age = 31;
```

Accessing Members of Structures

- structure member operator (.) or dot operator
  - accesses a structure member via the structure variable name

```
STUDENT myStudent;
printf( "%s", myStudent.age );
```

- structure pointer operator (->) or arrow operator
  - accesses a structure member via a pointer to the structure

```
STUDENT *myStudentPtr = &myStudent;
printf( “%d", myStudentPtr->age );
```

Structure Definition with *typedef*

- To avoid repeating ‘struct’ every time a variable is declared, use typedef

```
typedef struct {
        char name[10];
        char rollNum[10];
        int age;
  } STUDENT;

STUDENT myStudent, students[60];
```

Structures and Operators

- 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

Using Structures With Functions

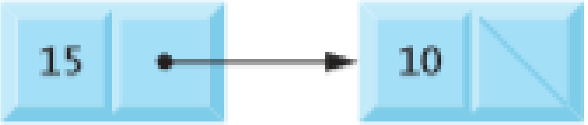
- Structures can be passed as arguments to functions
  - Entire structure or members of the structure

- Passing structures to functions
  - call by value
    - Pass entire structure
    - Or, pass individual members
  - call by reference
    - Pass structure variable’s address

Self Referential Structures

- A structure containing a member that is a pointer to the same structure type

```
struct Node{
    int data;
     struct Node *nextPtr;
};
```



  - A structure of type struct node has two members—integer member data and pointer member nextPtr
  - Member nextPtr points to a structure of type struct node
  - Member nextPtr is referred to as a link—i.e., nextPtr can be used to “tie” a structure of type struct node to another structure of the same type

Pointers and Structures

- Consider the following declaration:

```
struct inventory{
        char name[30];
        float price;
} product[2], *ptr;
```

- The assignment

  ptr = product; // would assign the address of the zeroth element of product to ptr

- Its members can be accessed using the following notation

  ptr -> name, ptr->price

Pointers and Structures contd.

- Initially the pointer ptr will point to product[0]
- when the pointer ptr is incremented by one it will point to next record, that is product[1]
- We can also use the notation

  (*ptr).number

  to access the member number
- ++ptr -> price;
  - Increments price, not ptr
- (++ptr) -> price;
  - increments ptr first and then links price

Unions

- Derived data type
- Members of a union share the same storage space
  - Each member within a structure is assigned its own memory location
- Size allotted is the size of largest member
- Members can be of any type, but only one data member can be referenced at a time
- Only the last data member's value can be accessed

Union - Definition and Declaration

```
union TempArea{
   int xInt;
   float xFloat;
};
union TempArea myVariable;
```

- TempArea is a union with 2 members
- myVariable is a variable of type TempArea

Union - Example

```
    /* number union definition */
    union number {
        int x;
      double y;
    }; /* end union number */

int main(){
    union number value;
    value.x = 100; /* put an
    integer into the union */
   printf( "\n int: %d \n double :
          %f",value.x, value.y );
```

```
    value.y = 100.0; /* put a
    double into the same union */
    printf( "\n int : %d \n float :
    %f", value.x,   value.y );
    return 0; /* indicates
    successful termination */
}
```

```
Output:
int:  100
double:  -
925595921174331360000000.000000
int:  0
double:  100.000000
```

Summary

- Structures are constructs used when related data of different type must be handled in a program
- Structures can be defined using ***struct*** keyword
- ***typedef*** keyword allows a structure to be used like normal data types, without the ***struct*** prefix
- ***sizeof*** a structure variable is at least the sum of sizes of all variables
- Unions are similar to structures and are used when one memory is used to store multiple types of values

Further Reading

Kernighan, B. W. and Richie, D. (1992) *The C Programming Language.* 2nd ed., New Delhi:PHI.