C PROGRAMING

Ketan Kore

Sunbeam Infotech

Structure

- Structure is a user-defined data type.
- Structure stores logically related (similar or non-similar) elements in contiguous memory location.
- Structure members can be accessed using "." operator via struct variable.
- Structure members can be accessed using "->" operator via struct pointer.
- Size of struct = Sum of sizes of struct members.
- If struct variable initialized partially at its point of declaration, remaining elements are initialized to zero.

```
// struct data-type declaration (global or local)
struct emp {
    int empno;
    char ename[20];
    double sal;
};
// struct variable declaration
struct emp e1 = {11, "John", 20000.0};
// print struct members
printf("%d%s%lf", e1.empno, e1.ename, e1.sal);
```

Struct – User defined data-type

- int a = 10;
- printf("%d", a);
- scanf("%d", &a);
- int *p = &a;
- printf("%d", *p);

- fun1(a);
- void fun1(int x) { … }
- fun2(&a);
- void fun1(int *p) { … }
- int arr[3] = {11, 22, 33};
- for(int i=0; i<3; i++)
  printf("%d", arr[i]);

- struct emp e = { 11, "John" };
- printf("%d, %s", e.empno, e.ename);
- scanf("%d%s", &e.empno, &e.ename);
- struct emp *p = &e;
- printf("%d, %s", p->empno, p->ename);
- printf("%d, %s", (*p).empno, (*p).ename);
- fun1(e);
- void fun1(struct emp x) { … }
- fun1(&e);
- void fun2(struct emp *p) { … }
- struct emp arr[3] = { {…}, {…}, {…} };
- for(int i=0; i<3; i++)
  printf("%d", arr[i].empno);

Struct

- A variable of a struct can be member of another struct.
- This can be done with nested struct declaration.

```
struct emp {
    int empno;
    char ename[20];
    double sal;
    struct {
        int day, month, year;
    }join;
};
```

```
struct date {
    int day, month, year;
};
struct emp {
    int empno;
    char ename[20];
    double sal;
    struct date join;
};
struct emp e = { 11, "John", 2000.0, {1, 1, 2000} };
printf("%d %s %d-%d-%d\n", e.empno, e.ename, e.join.day, e.join.month, e.join.year);
```

Struct padding

- For efficient access compiler may add hidden bytes into the struct called as "struct padding" or "slack bytes".
- On x86 architecture compiler add slack bytes to make struct size multiple of 4 bytes (word size).
- These slack bytes not meant to be accessed by the program.
- Programmer may choose to turn off this feature by using #pragma.
  - #pragma pack(1)

```
struct test {
    int a;
    char b;
};
printf("%u\n", sizeof(struct test));


#pragma pack(1)
struct test {
    int a;
    char b;
};
printf("%u\n", sizeof(struct test));
```

Bit Fields

- A bit-field is a data structure that allows the programmer to allocate memory to structures and unions in bits in order to utilize computer memory in an efficient manner.
- Bit-fields can be signed or unsigned.
  - Signed bit-field, MSB represent size + or -.
  - Unsigned bit-field, all bits store data.
- Limitations of bit-fields
  - Cannot take address of bit-field (&)
  - Cannot create array of bit-fields.
  - Cannot store floating point values.

```
struct student {
    char name[20];
    unsigned int age: 7;
    unsigned int roll: 6;
};
struct student s1 = { "Ram", 10, 21 };
printf("%s, %d, %d", s1.name, s1.age, s1.roll);
```

Union

- Union is user defined data-type.
- Like struct it is collection of similar or non-similar data elements.
- All members of union share same memory space i.e. modification of an member can affect others too.
- Size of union = Size of largest element
- When union is initialized at declaration, the first member is initialized.
- Application:
  - System programming: to simulate register sharing in the hardware.
  - Application programming: to use single member of union as per requirement.

File IO

- File is collection of data and information on storage device.
- Each file have data (contents) and metadata (information).
- File IO can enable read/write file data.
- File Input Output
  - Low Level File IO
    - Explicit Buffer Management. Use File Handle.
  - High Level File IO
    - Auto Buffer Management. Use File Pointer.
    - Formatted (Text) IO
      - fprintf(), fscanf()
    - Unformatted (Text) IO
      - fgetc(), fputc(), fgets(), fputs()
    - Binary File IO
      - fread(), fwrite()

High Level File IO

- File must be opened before read/write operation and closed after operation is completed.
- FILE * fp = fopen("filepath", "mode"); – to open the file
  - File open modes:
    - w: open file for write. If exists truncate. If not exists create.
    - r: open file for read. If not exists, function fails.
    - a: open file for append (write at the end). If not exists create.
    - w+: Same as "w" + read operation.
    - r+: Same as "r" + write operation.
    - a+: Same as "a" + append (write at the end) operation.
  - Return FILE* when opened successfully, otherwise return NULL.
- fclose(fp);
  - Close file and release resources.

File IO

- Character IO
  - fgetc()
  - fputc()
- String (Line) IO
  - fgets()
  - fputs()
- Formatted IO
  - fscanf()
  - fprintf()
- Binary (record) IO
  - fread()
  - fwrite()
- File position
  - fseek()
  - ftell()

Thank you!

Ketan Kore <ketan.kore@sunbeaminfo.com>