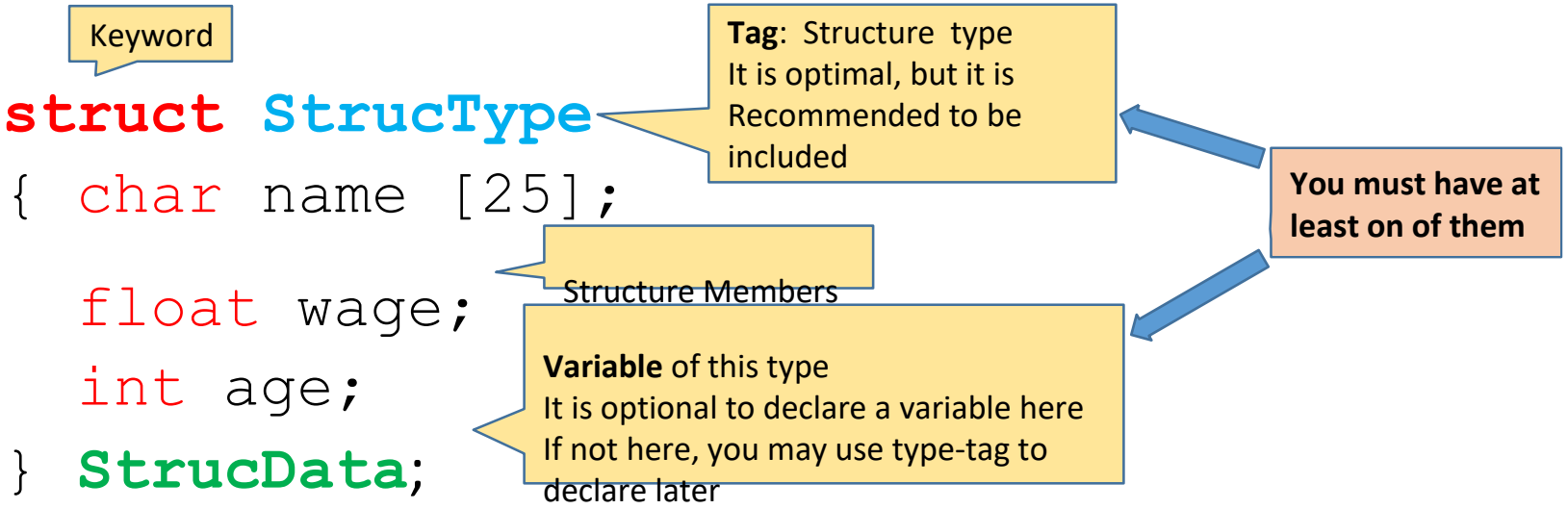
CS50 Notes

Chapter 7

Structures

Arrays: Collection of homogenous (same type) data

Structures: Collection of heterogeneous (different type) data




```
struct StrucType MoreStrucData;          // Used to define more data
struct StrucType StrucData1, StrucData2; // Used to define multiple data
```

Structure Type

- Recommended to include **structure type**.
- If structure type **not** included:
  - For each data the structure must be redefined.
  - Compiler might **not** recognize data defined by identical but separately structures, as identical structures.
- Examples:
  - Compiler might **not** recognize **Data1** and **Data2** to be the same type.
  - Compiler will recognize **Data3**, **Data4**, **Data5**, and **Data6** as the same type.

```
struct
 { int x;
   int y;
 } Data1;

struct
 { int x;
   int y;
 } Data2;
```

preferred

```
struct StType
 { int x;
   int y;
 } Data3;
struct StType Data4, Data5;
struct StType Data6;
(recommended)
```

Structure Components (Members)

- Structure may be initialized:

```
struct StType data1 = {4, 7};
```

- Structure may be partially initialized. In this case non-initialized members will be zero

```
struct StType data1 = {4};
```

- Structure member **cannot** be a function.
- Structure member **cannot** have instance of structure itself.
- Structure member **cannot** be a void type.
- Structure member **can be a pointer** to structure itself.
- Structure member **can be another** structure.
- Can have an array of structures.
- Structure member can be an array.
- Structure members can be referred by period notations.

```
data1.x = 34;
```

- Structure members can be used anywhere that type of variable can be used.

Structures - Examples

Examples:

```
struct StRecType
{     char fullName [25];
      int  scores  [4];
      int total;
      char grade;
} ;
struct StRecType oneRecord;                    // Single structure
struct StRecType Records [10];                 // Array of structures
Struct StRecType * ptrRecord;                  // Pointer to structure
//   important note:
//   you always have to assign address
//   of structure to ptrRecord before use.
oneRecord.scores [2] = 20;
char c = records[7].fullName[0];
(*ptrRecord).total
```

alternative notation: `ptrRecord->Total`

```
char c = ptrRecord->fullName[0];
if (records[7].fullName[0] == 'Z')
```

Pointers to Structures

Examples: Pointer to structure

```
struct StRecType
{     char fullName [25];
      int  scores  [4];
      int total;
      char grade;
} ;
struct StRecType * ptrToStruct;
(*ptrToStruct).grade  ➔ ptrToStruct->grade
```

Examples: Pointer as a member to structure itself

```
struct StRecType
{     char fullName [25];
      int  scores  [4];
      int total;
      char grade;
    struct StRecType * ptrToItslef;
} ;
```

Structures as Parameters to Functions

- Pointers to Structure are passed as parameters to functions
- Structure itself can be passed to function (call by value)
- Structure may return pointer or structure

Functions atof, atoi

- **`atof (string)`**, convert floating point number from string to float (ASCII to float)

  `atof ("1234.56")` returns **`123.45`**

atof stands for **ASCII** to **float**

- **`atoi (string)`** convert integer number from string to float (ASCII to integer)

  `atoi ("123456")` returns **`12345`**

atoi stands for **ASCII** to **integer**

- **`fgets ()`** Reads of all of text through newline.
  - Then you use the above function to get the number
- **`scanf ()`** Reads numeric text, until it finds a non-numeric character, converts numeric text to number
  - No need to use atof() and atoi()

Enumerated Types

- It is used for readability
- Declares a set of integer constants

  `enum boolean {false, true};` is almost equivalent to:

  ```
  #define false      0
  #define true       1
  ```

- Then you define variable.

  ```
  enum boolean answer;
  ```

- `answer` can only accepts false or true
- Another example:

  ```
  enum weekdays {Sun, Mon, Tue, Wed, Thu, Fri, Sat};
  ```

- Their values are 0, 1, 2,… , unless you change them:

  ```
  enum weekdays {Sun =9 , Mon, Tue, Wed = 7, Thu = 7, Fri, Sat};
  ```

- Their values now are, respectively 9, 10, 11, 7, 7, 8, 9
- Cannot change their values after declaration and initialization

  `Mon = 5` is illegal

Enumerated Types

Example: This example shows the readability of using **enum**.

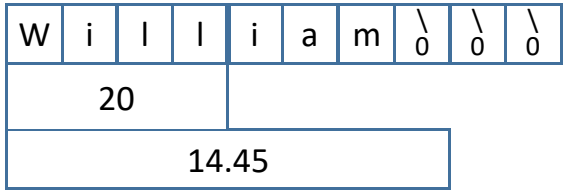
```
enum weekdays {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
enum weekdays dayOfWeek;
char sign [50];

switch (dayOfWeek)
{    case Sunday:       strcpy (sign, "Closed");
                        break;
     case Monday:       strcpy (sign, "8:00 AM – 8:00 PM");
                        break;
     case Tuesday:      strcpy (sign, "8:00 AM – 8:00 PM");
                        break;
     case Wednesday:    strcpy (sign, "8:00 AM – 8:00 PM");
                        break;
     case Thursday:     strcpy (sign, "8:00 AM – 8:00 PM");
                        break;
     case Friday:       strcpy (sign, "8:00 AM – 6:00 PM");
                        break;
     case Saturday:     strcpy (sign, "9:00 AM – 6:00 PM");
                        break;
}
```

Unions (brief review suffices)

Similar to structures except the members overlapped.

```
struct StType
{ char name [10];
  int score;
  double wage;
} data1 = {"William", 20, 14.45};
```



Union:

```
union UType
{ char name [10];
  int score;
  double wage;
} data1 ;

data1.strcpy (name, "William");
data1.score = 20;
data1.wage = 14.45;
```

Bit Fields

- Skip

CS50 Notes – Last Slide Chapter 7