CSCI 2170 Lecture Notes on typedef, enum, and struct types

---

typedef is used to create an alias of a type (C++ built in type or any user defined type). It is used to make the program easy to read and modify.

Example

(1)
```
typedef    unsigned int    myintType;
myintType   age;
```

(2)
```
typedef    float        balanceType;
balanceType   mutualFund;
```

(3)
```
typedef    char  wordType[WORD_LENGTH];
wordType    selectedWord;
```

(3)
```
typedef    char   wordListType[MAX_WORDS][WORD_LENGTH];
wordListType   wordBank;
```

---

enumeration type (enum)

A user defined data type whose domain is an ordered set of literal values expressed as identifiers.

Examples:

(1)
```
enum   Days  {SUN, MON, TUE, WED, THU, FRI, SAT};
```

notes: the identifiers are ordered : SUN < MON < TUE … < SAT
the default values for the identifiers are: SUN=0, MON=1, …SAT=6, (but the values can be changed if necessary)

(2)
```
enum Vowel {'A', 'E', 'I', 'O', 'U'};    //  wrong!!  Why?
```

(3)
```
enum  Animals  {CAT, DOG, BIRD, HORSE, SHEEP, TIGER, LION};
Animals   firstAnimal, secondAnimal, thirdAnimal;

// assignment statements
firstAnimal = CAT;
secondAnimal = DOG;
thirdAnimal = firstAnimal;
firstAnimal = 0;   //wrong!
secondAnimal = 30;   // wrong!

// increment
firstAnimal = Animals(firstAnimal + 1);
```

enum used in switch statement:

```
switch  (firstAnimal)
{
case   CAT:    …
               break;
case   DOG:    …
               break;
case   BIRD:    …
               break;
case   HORSE: …
                break;
case   SHEEP: …
                break;
case    LION:   …
                break;
case    TIGER: …
                break;
}
```

enum used in array subscripts

(1)
```
Animals     oneAnimal;
float          weights[7];

for   (oneAnimal = CAT;   oneAnimal <=TIGER;    oneAnimal =
Animals(oneAnimal+1))
        cout   << "The average weight for this animal is " << weights[oneAnimal]
<< endl;
```

---

struct – a user defined, structural type
Used to define a record consisting of a collection of members of different types

- declaration

```
struct    employee
{
    int       id;
    string  name;
    char     gender;
    int       numOfDependents;
    float     payByMonth[12];
};

typedef    struct employee employeeType;

employeeType   teacher1, teacher2;
```

- **members of a struct are accessed using '.' notation**

```
teacher1.id = 3222;
teacher1.name = "Lori Weinberg";
teacher1.gender = 'f';
teacher1.numOfDependents = 2;
for (int i=0; i<12; i++)
      teacher1.payByMonth[i] = 3000;
```

- **member of a struct type can be another struct type, BUT, it can not have a member of the type currently being defined, i.e., no circular definition**

```
    struct   date
    {
             int month;
             int day;
             int year;
    };
    typedef struct date dateType;
```

    add it to the employeeType as a member:

```
            struct    employee
            {
                     int       id;
                     char      name[MAX_LENGTH];
                     char      gender;
                     int       numOfDependents;
                     float     payByMonth[12];

                     dateType        dateStarted;
                     dateType        dateTerminated;
            };
```

```
    // The following definition is wrong:  ➔
            struct    employee
            {
                     int       id;
                     string    name;
                     char      gender;
                     int       numOfDependents;
                     float     payRate;

                     employee   bestFriend;      // recursive definition is not allowed!!!
            }
```

‘.’ Notation is used to access member of struct variable, apply right association

```
employee teacher1;
teacher1.dateStarted.month = 8;
teacher1.dateStarted.day = 1;
teacher1.dateStarted.year = 1999;
teacher1.dateTerminated.month = 7;
teacher1.dateTerminated.day = 31;
teacher1.dateTerminated.year=2002;
```

- output individual members of a structural variable

  ```
  cout << “Employee “ << teacher1.id << “: “ << endl;
  cout << “Name :      “ << teacher1.name << endl;
  ```

- **assign structural variable as a whole (aggregated assignment)**, each member of one record is assigned to the corresponding member of the other record.

  ```
  employee teacher2;
  teacher2 = teacher1;   ← do not have to assign each member separately
  ```

- **compare two variables of the same structural type**

  ```
  // Not Allowed ➔      if  (teacher1 == teacher2)
                               ….
  // Correct ➔      if ((teacher1.id == teacher2.id) &&
                         !(strcmp(teacher1, teacher2) &&
                         (teacher1.gender == teacher2.gender) &&
                         (teacher1.numOfDependents == teacher2.numOfDependents))
                         {
                                  …
  ```

- **passing struct variable to function**

```
void ReadData(employeeType &);                       // by reference
// void PrintData(employeeType );                    // by value
void PrintData (const  employeeType &);           // const reference

int main()
{
    employeeType    teacher, chairman, dean;

    ReadData (chairman);
    ReadData(teacher);
    ReadData(dean);
     …
    PrintData(teacher);
}

void ReadData(employeeType & emp)
```

```
{
        cout << “Enter the employee’s id”<< endl;
        cin >> emp.id;
        cout << “The employee id entered is “ << emp.id << endl;

        cout << “Enter the employee’s name” << endl;
        cin >> emp.name;
        cout << “The employee’s name is “ << emp.name << endl;

        …
        return;
}

void   PrintData(const employeeType& emp)
{
        cout << setw(10) << emp.id << setw(20) <<emp.name ;
        cout << setw(10) << emp.payRate*20 << endl;

        return;
}
```