Lecture 8.2: Overview

- Data Structures
  - Structures
    - Declaration and definition
    - Instantiation and initialization
    - Member access
  - Unions
    - Declaration and definition
    - Member access
  - Enumerators
    - Declaration and definition
  - Type definitions
  - Examples
    - Student records: **`Students.c`**

Data Structures

- Structures (aka. records): **`struct`**
  - User-defined, composite data type
    - Type is a composition of (different) sub-types
  - Fixed set of members
    - Names and types of members are fixed at structure definition
  - Member access by name
    - Member-access operator: ***`structure_name.member_name`***
- Example:

```
struct S { int i; float f;} s1, s2;

s1.i = 42;      /* access to members */
s1.f = 3.1415;
s2 = s1;        /* assignment */
s1.i = s1.i + 2*s2.i;
```

Data Structures

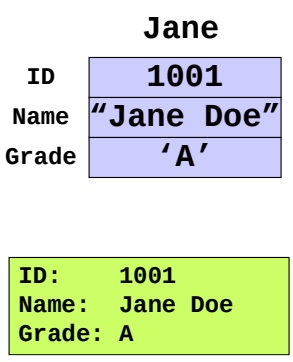
- Structure Declaration
  - Declaration of a user-defined data type
- Structure Definition
  - Definition of structure members and their type
- Structure Instantiation and Initialization
  - Definition of a variable of structure type
  - Initializer list defines initial values of members
- Example:

```
struct Student;          /* declaration */

struct Student           /* definition */
{ int   ID;              /* members */
  char  Name[40];
  char  Grade;
};

struct Student Jane =    /* instantiation */
{1001, "Jane Doe", 'A'}; /* initialization */
```


Data Structures

- Structure Access
  - Members are accessed by their name
  - Member-access operator `.`
- Example:

```
struct Student
{  int  ID;
   char Name[40];
   char Grade;
};

struct Student Jane =
{1001, "Jane Doe", 'A'};

void PrintStudent(struct Student s)
{
   printf("ID:    %d\n", s.ID);
   printf("Name:  %s\n", s.Name);
   printf("Grade: %c\n", s.Grade);
}
```




```
ID:    1001
Name:  Jane Doe
Grade: A
```

Data Structures

- Unions: **`union`**
  - User-defined, composite data type
    - Type is a composition of (different) sub-types
  - Fixed set of *mutually exclusive* members
    - Names and types of members are fixed at union definition
  - Member access by name
    - Member-access operator: ***`union_name.member_name`***
  - *Only one member may be used at a time!*
    - *All members share the same location in memory!*
- Example:

```
union U { int i; float f;} u1, u2;

u1.i = 42;     /* access to members */
u2.f = 3.1415;
u1.f = u2.f;   /* destroys u1.i! */
```


Data Structures

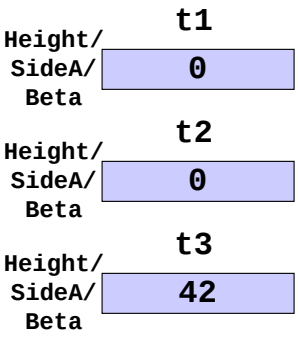
- Union Declaration
  - Declaration of a user-defined data type
- Union Definition
  - Definition of union members and their type
- Union Instantiation and Initialization
  - Definition of a variable of union type
  - *Single* initializer defines value of *first* member
- Example:

```
union HeightOfTriangle;  /* declaration */

union HeightOfTriangle   /* definition */
{ int   Height;          /* members */
  int   LengthOfSideA;
  float AngleBeta;
};

union HeightOfTriangle H /* instantiation */
= { 42 };                /* initialization */
```

Data Structures

- Union Access
  - Members are accessed by their name
  - Member-access operator `.`
- Example:

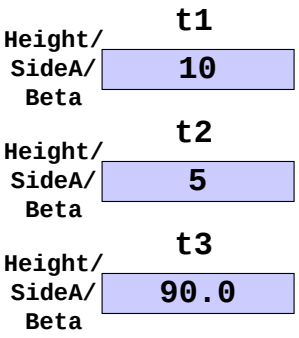
```
union HeightOfTriangle
{ int   Height;
  int   SideA;
  float Beta;
};

union HeightOfTriangle t1, t2, t3
= { 42 };
```



Data Structures

- Union Access
  - Members are accessed by their name
  - Member-access operator `.`
- Example:

```
union HeightOfTriangle
{ int   Height;
  int   SideA;
  float Beta;
};

union HeightOfTriangle t1, t2, t3
= { 42 };

void SetHeight(void)
{
   t1.Height = 10;
   t2.SideA = t1.Height / 2;
   t3.Beta = 90.0;
}
```



Data Structures

- Enumerators: **enum**
  - User-defined data type
    - Members are an enumeration of integral constants
  - Fixed set of members
    - Names and values of members are fixed at enumerator definition
  - Members are constants
    - Member values cannot be changed after definition
- Example:

```
enum E { red, yellow, green };
enum E LightNS, LightEW;

LightEW = green;        /* assignment */
if (LightNS == green)   /* comparison */
   { LightEW = red; }
```


Data Structures

- Enumerator Declaration
  - Declaration of a user-defined data type
- Enumerator Definition
  - Definition of enumerator members and their value
- Enumerator Instantiation and Initialization
  - Definition of a variable of enumerator type
  - Initializer should be one member of the enumerator
- Example:

```
enum Weekday;             /* declaration */

enum Weekday              /* definition */
{ Monday, Tuesday,        /* members */
  Wednesday, Thursday,
  Friday, Saturday, Sunday
};

enum Weekday Today        /* instantiation */
= Thursday;               /* initialization */
```

Data Structures

- Enumerator Values
  - Enumerators are represented as integer constants
  - By default, enumerator values start at 0 and are incremented by 1 for each following member
- Example:

```
enum Weekday
{ Monday,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday,
  Sunday
};
enum Weekday Today
= Thursday;
void PrintWeekday(
     enum Weekday d)
{
  printf("Day: %d\n", d);
}
```

```
Today
Thursday

Day: 3
```


Data Structures

- Enumerator Values
  - Enumerators are represented as integer constants
  - By default, enumerator values start at 0 and are incremented by 1 for each following member
  - Specific enumerator values may be defined by the user
- Example:

```
enum Weekday
{ Monday = 1,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday,
  Sunday
};
enum Weekday Today
= Thursday;
void PrintWeekday(
     enum Weekday d)
{
  printf("Day: %d\n", d);
}
```

```
Today
Thursday

Day: 4
```

Data Structures

- Enumerator Values
  - Enumerators are represented as integer constants
  - By default, enumerator values start at 0 and are incremented by 1 for each following member
  - Specific enumerator values may be defined by the user
- Example:

```
Today
Thursday

Day: 5
```

```
enum Weekday
{ Monday = 2,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday,
  Sunday = 1
};

enum Weekday Today
= Thursday;

void PrintWeekday(
     enum Weekday d)
{
  printf("Day: %d\n", d);
}
```


Data Structures

- Type definitions: **`typedef`**
  - A *typedef* can be defined as an alias type for another type
  - A *typedef* definition follows the same rules as a variable definition
  - Type definitions are usually used to abbreviate access to user-defined types
- Examples:

```
typedef signed long int MyInteger;
MyInteger i1 = 42;

typedef enum Weekday Day;
Day Today = Thursday;

typedef struct Student Scholar;
Scholar Jane, John;
```

Data Structures

- Program example: **`Students.c`** (part 1/6)

```
/* Students.c: simple array of student records  */
/* author: Rainer Doemer                        */
/* modifications:                               */
/* 08/28/13 RD initial version                  */

#include <stdio.h>
#include <stdlib.h>

/* constants */

#define SLEN 40
#define MAX 100

/* data structures */

struct Student
{  int  ID;
   char Name[SLEN];
   int  Score;
};
typedef struct Student STUDENT;

STUDENT Record[MAX];
int N = 0;
...
```


Data Structures

- Program example: **`Students.c`** (part 2/6)

```
...
/* function declarations */

STUDENT EnterStudent(void);
void PrintStudent(STUDENT s);
char LetterGrade(int Score);
void InsertStudent(STUDENT s);

/* function definitions */

STUDENT EnterStudent(void)
{
    STUDENT s;

    printf("Enter student ID:    ");
    scanf("%d", &s.ID);
    printf("Enter student name:  ");
    scanf("%39s", &s.Name[0]);
    printf("Enter student score: ");
    scanf("%d", &s.Score);
    return s;
}

...
```

Data Structures

- Program example: **`Students.c`** (part 3/6)

```
...
void PrintStudent(STUDENT s)
{
    printf("ID %3d: %-39s, Score %3d%% = %c\n",
                s.ID, s.Name, s.Score, LetterGrade(s.Score));
}

char LetterGrade(int Score)
{   switch(Score/10)
    {   case 10:
        case  9: return 'A';
        case  8: return 'B';
        case  7: return 'C';
        case  6: return 'D';
        case  5: case 4: case 3: case 2: case 1:
        case  0: return 'F';
        default: break;
    } /* hctiws */
    return '-';
}
...
```


Data Structures

- Program example: **`Students.c`** (part 4/6)

```
...

void InsertStudent(STUDENT s)
{
    int i, j;

    for(i=0; i<N; i++)
    {
        if (s.ID < Record[i].ID)
        { break;
        }
    }
    for(j=N; j>i; j--)
    {
        Record[j] = Record[j-1];
    }
    Record[i] = s;
    N++;
}

...
```

Data Structures

- Program example: **`Students.c`** (part 5/6)

```
...

int main(void)
{
    int Choice, i;

    while(1)
    {   printf("Student records: %d\n", N);
        printf("1. Enter new student\n");
        printf("2. Print student table\n");
        printf("3. Quit\n");
        printf("Choice: ");
        scanf("%d", &Choice);
        switch(Choice)
        { case 1:
            {   if (N < MAX)
                {   InsertStudent(EnterStudent());
                }
                break;
            }
...
```


Data Structures

- Program example: **`Students.c`** (part 6/6)

```
...

          case 2:
            {   for(i=0; i<N; i++)
                { PrintStudent(Record[i]);
                }
                break;
            }
          case 3:
            {   exit(0);
            }
          default:
            {   break;
            }
        } /* hctiws */
    } /* elihw */
    return 0;
} /* end of main */

/* EOF */
```

Data Structures

- Example session: **`Students.c`** (part 1/3)

```
% vi Students.c
% gcc Students.c -o Students -Wall -ansi
% Students
Student records: 0
1. Enter new student
2. Print student table
3. Quit
Choice: 1
Enter student ID:    1879
Enter student name:  Albert_Einstein
Enter student score: 100
Student records: 1
1. Enter new student
2. Print student table
3. Quit
Choice: 2
ID 1879: Albert_Einstein               , Score 100% = A
...
```


Data Structures

- Example session: **`Students.c`** (part 2/3)

```
...
Student records: 1
1. Enter new student
2. Print student table
3. Quit
Choice: 1
Enter student ID:    1642
Enter student name:  Isaac_Newton
Enter student score: 100
Student records: 2
1. Enter new student
2. Print student table
3. Quit
Choice: 2
ID 1642: Isaac_Newton                  , Score 100% = A
ID 1879: Albert_Einstein               , Score 100% = A
...
```

Data Structures

- Example session: **`Students.c`** (part 3/3)

```
[...]
Choice: 1
Enter student ID:    1623
Enter student name:  Blaise_Pascal
Enter student score: 100
Student records: 3
1. Enter new student
2. Print student table
3. Quit
Choice: 2
ID 1623: Blaise_Pascal                  , Score 100% = A
ID 1642: Isaac_Newton                   , Score 100% = A
ID 1879: Albert_Einstein                , Score 100% = A
Student records: 3
1. Enter new student
2. Print student table
3. Quit
Choice: 3
%
```