CME 112- Programming Languages II

Week 8
Enum, Typedef and Struct

Assist. Prof. Dr. Caner Özcan

Science is trying to understand the language of nature. Those who understand the language are friendly to nature, and those who do not understand are enemies.
An enumeration is a user-defined data type that consists of integral constants. To define an enumeration, keyword `enum` is used.

- Enumeration constants are like symbolic constants.
  - Enum values are set automatically
  - Values start at 0 and incremented by 1
  - Need unique constant names

We can create our own data type by `enum`

For example we can create a new boolean type in which 0 is false and 1 is true
#include<stdio.h>
int main( void )
{
    // Define new data type boolean
    enum boolean {
        false = 0,
        true = 1
    };
    // Now define a variable with new data type boolean
    enum boolean isTrue;
    isTrue = true;
    if( isTrue == true )
        printf( "True\n" );
    return 0;
}
Enum

Enum in C

<table>
<thead>
<tr>
<th>Declaration</th>
<th>enum variable</th>
<th>state=0</th>
<th>state=1</th>
<th>state=6</th>
</tr>
</thead>
<tbody>
<tr>
<td>enum</td>
<td>days-of-week</td>
<td>{ Sun, Mon, Tue, Wed, Thu, Fri, Sat }</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instantiation

enum days-of-week day;

Operation

day = wed;  day = 2

Enumerators
(list of constants separated by commas)

Object of enum days-of-week
As state of wed=2
```c
#include<stdio.h>
int main( void )
{
    // Define new data type mainColors
    enum mainColors {
        Red,
        Blue,
        Yellow
    };

    // Define variable
    enum mainColor pixel;

    // Set value of pixel to blue
    // You can set Yellow or Red also.
    pixel = Blue;

    // Compare variable's value.
    if( pixel == Red )
        printf( "Red pixel \n" );
    else if( pixel == Blue )
        printf( "Blue pixel \n" );
    else
        printf( "Yellow pixel\n" );

    return 0;
}
```
// An example program to demonstrate working of enum in C
#include<stdio.h>

enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};

int main()
{
    enum week day;
    day = Wed;
    printf("%d",day);
    return 0;
}
Two enum names can have same value. For example, in the following C program both ‘Failed’ and ‘Freezed’ have same value 0.

```
#include <stdio.h>
enum State {Working = 1, Failed = 0, Freezed = 0};

int main()
{
    printf("%d, %d, %d", Working, Failed, Freezed);
    return 0;
}
```
#include <stdio.h>
enum day {sunday = 1, monday, tuesday = 5, wednesday, thursday = 10, friday, saturday};

int main()
{
    printf("%d %d %d %d %d %d %d", sunday, monday, tuesday, wednesday, thursday, friday, saturday);
    return 0;
}

Output: 1 2 5 6 10 11 12
Enum

Enum is useful if you have some data to be grouped or arranged. Some examples:

- enum education { primarySchool, secondarySchool, highSchool, graduate, master };
- enum education student;
- enum sex { male, female };
- enum sex person;

Notice that we use enum keyword for each variable definition.

We have two alternative ways for not writing enum for each variable definition

- Defining variable with enum definition
- Using typedef
#include<stdio.h>
int main( void )
{
    // Define new data type
    // Also define a new variable with the new data type,
    
    enum boolean {
        false = 0,
        true = 1
    } isTrue;
    
    isTrue = true;
    if( isTrue == true )
        printf( "True \n" );
    return 0;
}
Typedef

- Used to name data types with other user defined names.

- Format of typedef:
  - typedef old_datatype_name new_datatype_name
  - typedef int tamsayi
    - defines int as tamsayi
```c
#include<stdio.h>
int main( void )
{
    // Define new data type
    // Also define a new variable with the new data type,
    enum boolean {
        false = 0,
        true = 1
    };
    // With this definition we can create boolean type variables with
    // one step
    typedef enum boolean bool;

    bool isTrue;

    isTrue = true;
    if( isTrue == true )
        printf( "True \n" );
    return 0;
}
```
If enum is defined globally it can be used as parameter to a function.
void writeMonthName( months nameOfMonth )
{
    switch( nameOfMonth ) {
    case january: printf( "January\n" );break;
    case february: printf( "February\n" );break;
    case march: printf( "March\n" );break;
    case april: printf( "April\n" );break;
    case may: printf( "May\n" );break;
    case june: printf( "June\n" );break;
    case july: printf( "July\n" );break;
    case august: printf( "August\n" );break;
    case september: printf( "September\n" );break;
    case october: printf( "October\n" );break;
    case november: printf( "November\n" );break;
    case december: printf( "December\n" );break;
    }
}
Struct

- Used to group different types of variables in one structure.
- Structures are important for object oriented programming

```c
#include<stdio.h>
int main( void )
{
    struct {
        int year;
        int month;
        int day;
    } birth_day;
    printf( "Enter your birth day " );
    printf( " in MM-DD-YYYY format> ");
    scanf( "%d-%d-%d", &birth_day.month,
           &birth_day.day,
           &birth_day.year );
    printf( "Your birth day: " );
    printf( "%d/%d/%d\n", birth_day.month,
            birth_day.day,
            birth_day.year );

    return 0;
}
```
If we did not use struct in this sample we would have to define 9 different variables.

We make it with 3 variables.

Think about a program that takes 20 information for a person.

With 3 person you have to define 60 different variables.

Another advantage of structures is to easily copy data from one variable to another.

you = yourSister assignment copies your sister's data on to your data.
#include<stdio.h>
int main( void )
{
    struct {
        char name[40];
        int length;
        struct {
            int year;
            int month;
            int day;
        } bornInformation;
    } person;
    printf( "Your name: " );
    scanf( "%s", person.name );
    printf( "Your length: " );
    scanf( "%d", &person.length );
    printf( "Your birth day: " );
    scanf( "%d-%d-%d", &person.bornInformation.month, &person.bornInformation.day, &person.bornInformation.year );
    printf( "Entered information: \n" );
    printf( "Name: %s\n", person.name );
    printf( "Length: %d\n", person.length );
    printf( "Birth day: %d/%d/%d\n", person.bornInformation.month, person.bornInformation.day, person.bornInformation.year );

    return 0;
}
Labeling Structures

- Labeling structures has many advantages.
- If labeling is not done, you have to define the variables when defining the structure.
- If you use labels you can define as many variables as you want from struct in any point of your program.
- In order to use the struct, we must create a variable with the label that we defined.

```c
#include<stdio.h>
#include<string.h>
int main( void )
{
    // personData is the label of our struct
    struct person_Data {
        char name[40];
        int length;
    };

    // We create two variables using struct.
    struct person_Data person_1;
    struct person_Data person_2;

    // We store the first person's data.
    strcpy( person_1.name, "AHMET" );
    person_1.length = 170;

    // We store the second person's data.
    strcpy( person_2.name, "MEHMET" );
    person_2.length = 176;
    return 0;
}
```
Initial Values for Structures

- Structures can be defined with initial values of its variables.
- Order of the values must fit to the struct's order.
- You can give initial values for structs defined with or without label.

```c
#include<stdio.h>
int main( void )
{
    struct {
        char name[40];
        int length;
    } person = { "Ali", 167 };
    return 0;
}
```
Arrays of Structures

```c
#include<stdio.h>
int main( void )
{
    int i;
    struct birthDate {
        int day;
        int month;
        int year;
    };
    struct person_data {
        char name[40];
        int length;
        //Define a variable of an other structure type inside
        //struct
        struct birthDate date;
    };
                                    "Veli", 178, { 14, 4, 1980 },
                                    "Cenk", 176, { 4, 11, 1983 } };

    // Print all values of people defined in array
    for( i = 0; i < 3; i++ ) {
        printf( "Record No.: %d\n", ( i + 1 ) );
        printf( "Name: %s\n", person[i].name );
        printf( "Length: %d\n", person[i].length );
        printf( "Birth Date: %d/%d/%d\n", person[i].date.day,
                person[i].date.month,
                person[i].date.year );
    }
    return 0;
}
```
Accessing Structures with Pointers

```c
#include<stdio.h>
int main( void )
{
    int i;

    struct birthDate {
        int day;
        int month;
        int year;
    };

    struct person_data {
        char name[40];
        int length;
        // Define a variable of an other structure type inside
        // struct
        struct birthDate date;
    };

    struct person_data *ptr;

                                    "Veli", 178, { 14, 4, 1980 },
                                    "Cenk", 176, { 4, 11, 1983 });

    // Print all values of people defined in array
    for( i = 0, ptr = &person[0]; ptr <= &person[2]; ptr++, i++ ) {
        printf( "Record No.: %d\n", ( i + 1 ) );
        printf( "Name: %s\n", ptr->name );
        printf( "Length: %d\n", ptr->length );
        printf( "Birth day: %d/%d/%d\n", ptr->date.day,
                ptr->date.month,
                ptr->date.year );
    }

    return 0;
}
```
Passing Structs as Parameters to Functions

➢ Define structure globally and pass to function.
Next Week

- Singly Linked Linear Lists
References


► Paul J. Deitel, “C How to Program”, Harvey Deitel.

► “A book on C”, All Kelley, İra Pohl
Thanks for listening

Any questions?