7.1 Grouping data: struct

Sometimes two data items are really aspects of the same data. For example, time might be recorded in hours and minutes, as in 4 hours and 23 minutes. Or a point on a plot might be recorded as x = 5, y = 7. Storing such data in separate variables, such as runTimeHours and runTimeMinutes, is not as clear as grouping that data into a single variable, like runTime, which might have subitems runTime.hours and runTime.minutes.

PARTICIPATION ACTIVITY 7.1.1: Naturally grouped data.

1) Select the pair forming part of a person's height (in U.S. units)

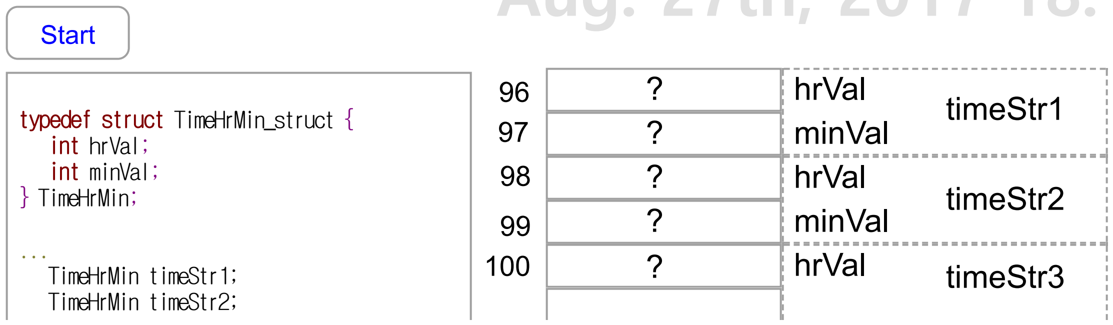
- ○ Feet and inches
- ○ Inches and salary
- ○ Pounds and ounces

2) Select the group of items indicating the change provided to a person who pays for a meal.

- ○ Ounce, gill, pint, quart, and gallon
- ○ Mile, furlong, yard, feet, and inches
- ○ Dollars, quarters, dimes, nickels, and pennies

The ***struct*** construct defines a new type, which can be used to declare a variable with subitems. The following animation illustrates.

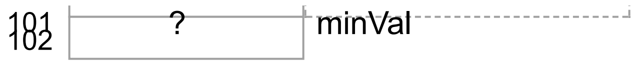
PARTICIPATION ACTIVITY 7.1.2: A struct enables creating a variable with data members.



```
typedef struct TimeHrMin_struct {
   int hrVal;
   int minVal;
} TimeHrMin;

...
   TimeHrMin timeStr1;
   TimeHrMin timeStr2;
   TimeHrMin timeStr3;

   timeStr1.hrVal = 5;
   timeStr1.minVal = 46;
   timeStr3.hrVal = timeStr1.hrVal;
```



The programmer uses struct to defines and use a new type as follows.

Construct 7.1.1: Defining and using a new struct type.

```
typedef struct StructTypeName_struct {
   type item1;
   type item2;
   ...
   type itemN;
} StructTypeName;


...
StructTypeName myVar;

myVar.item1 = ...
```

The above uses a common combination of a typedef definition with a struct definition. A ***typedef*** defines a new type name for an existing type. This material uses that combination exclusively and does not discuss typedef definition separately.

The `struct StructTypeName_struct { ... }` part defines a new struct type named `struct StructTypeName_struct`. The typedef part defines a new type name named `StructTypeName` that is synonymous with `struct StructTypeName_struct`.

A programmer can use StructTypeName to declare a variable of that struct type as in the statement `StructTypeName myVar;`.

Each `type` may be any type like int or char. Each struct subitem is called a ***data member***. For a declared variable, each struct data member can be accessed using ".", known as a ***member access*** operator, sometimes called ***dot notation***.

Assigning a variable of a struct type to another such variable automatically assigns each corresponding data member, as shown below.

PARTICIPATION ACTIVITY 7.1.3: Assigning variables of struct type assigns every corresponding data member.

Animation captions:

1. Assigning variables of struct type assigns every corresponding data member.

Forgetting to include the semicolon at the end of a struct definition will generate cryptic compilation errors:

Figure 7.1.1: Less-than-helpful error message when forgetting the semicolon at the end of a struct definition.

```
gcc -Wall testfile.c
testfile.c:6: error: two or more data types in declaration specifiers
testfile.c:6: warning: return type of 'main' is not 'int'
testfile.c: In function 'main':
testfile.c:7: error: incompatible types in return
testfile.c:8: warning: control reaches end of non-void function
```

Try 7.1.1: Internet search for clues of error message cause.

Do an Internet search by copying and pasting the following (from the second line of the above figure):

error: two or more data types in declaration specifiers

Then, read over the first 3 search results, particularly focusing on the reply messages to find clues to the error message's cause.

PARTICIPATION ACTIVITY 7.1.4: The struct construct.

1) A struct definition for CartesianPoint has subitems int x and int y. How many int locations in memory does the struct definition allocate?

Check Show answer

2) If struct definition CartesianPoint has subitems int x and int y, how many total int locations in memory are allocated

for these variable declarations?

```
int myNum;
CartesianPoint myPoint1;
CartesianPoint myPoint2;
```

Check Show answer

3) Given time1 is of type TimeHrMn defined earlier. What is the value of variable min after the following statements?

```
time1.hrVal  = 5;
time1.minVal = 4;
min = (60 * time1.hrVal) + time1.minVal;
```

Check Show answer

4) Write a statement to assign 12 to the hrVal data member of TimeHrMn variable time1.

Check Show answer

5) Write a statement that assigns the value of the hrVal data member of time1 into the hrVal data member of time2.

Check Show answer

6) Write a single statement that assigns the values of all data members of time1 to the corresponding data members of time2.

Check Show answer

7) Declare a variable person1 of type Person, where Person is already defined as a struct type.

Check Show answer

CHALLENGE ACTIVITY 7.1.1: Defining a struct.

Define a struct named PatientData that contains two integer data members named heightInches and weightPounds. Sample output for the given program:

```
Patient data: 63 in, 115 lbs
```

```
#include <stdio.h>

/* Your solution goes here  */

int main(void) {
   PatientData lunaLovegood;

   lunaLovegood.heightInches = 63;
   lunaLovegood.weightPounds = 115;

   printf("Patient data: %d in, %d lbs\n", lunaLovegood.heightInches, lunaLovegood.weightPoun

   return 0;
}
```

Run

CHALLENGE ACTIVITY 7.1.2: Accessing a struct's data members.

Write a statement to print the data members of InventoryTag. End with newline. Ex: if itemID is 314 and quantityRemaining is 500, print:

```
Inventory ID: 314, Qty: 500
```

```
#include <stdio.h>

typedef struct InventoryTag_struct {
   int itemID;
   int quantityRemaining;
} InventoryTag;

int main(void) {
   InventoryTag redSweater;

   redSweater.itemID = 314;
   redSweater.quantityRemaining = 500;

   /* Your solution goes here  */

   return 0;
}
```

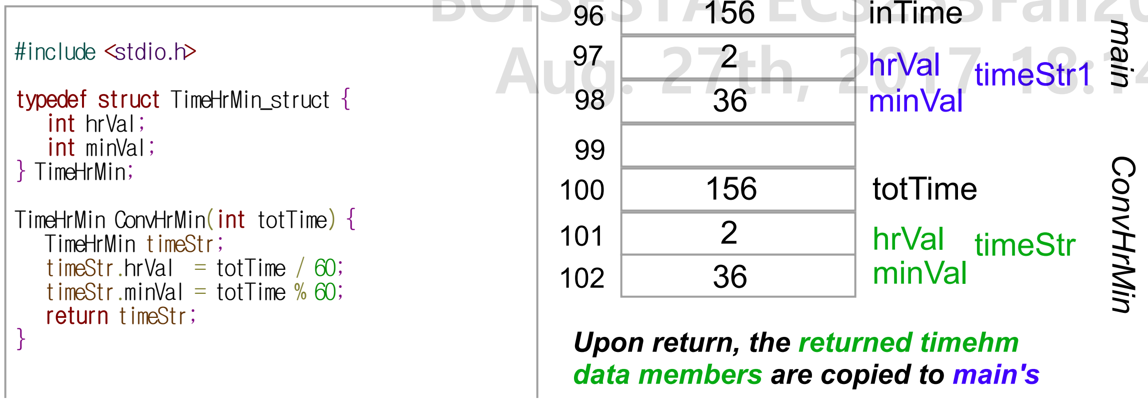
Run

7.2 Structs and functions

The struct construct's power is evident when used with functions. A struct can be used to return multiple values. The following illustrates. Although ConvHrMin() has two output values, the struct type allows the function to return a single item, avoiding a less-clear approach using two pass by reference parameters.

PARTICIPATION ACTIVITY 7.2.1: Using a struct that is returned from a function; the struct's data members are copied upon return.

Start animation



```
#include <stdio.h>

typedef struct TimeHrMin_struct {
   int hrVal;
   int minVal;
} TimeHrMin;

TimeHrMin ConvHrMin(int totTime) {
   TimeHrMin timeStr;
   timeStr.hrVal  = totTime / 60;
   timeStr.minVal = totTime % 60;
   return timeStr;
}
```

Upon return, the returned timehm data members are copied to main's

```
int main(void) {
   int inTime = 0;
   TimeHrMin timeStr1;

   printf("Enter tot minutes: ");
   scanf("%d", &inTime);
   timeStr1 = ConvHrMin(inTime);
   printf("Equals: ");
   printf("%d hrs ", timeStr1.hrVal);
   printf("%d mins\n", timeStr1.minVal);
   return 0;
}
```

timehm variable

```
Enter tot minutes:   156
Equals: 2  hrs   36  mins
```

PARTICIPATION ACTIVITY 7.2.2: Monetary change program.

Complete the program to compute monetary change, using the largest coins possible.

Load default template...

```

#include <stdio.h>

typedef struct MonetaryChange_struct {
   int quarters;
   // FIXME: Finish data members
} MonetaryChange;

MonetaryChange ComputeChange(int cents) {
   MonetaryChange change;

   // FIXME: Finish function
   change.quarters = 0; // FIXME

   return change;
}

int main(void) {
   int userCents = 0;
   MonetaryChange change;

```

```
119
```

Run

PARTICIPATION ACTIVITY 7.2.3: Functions returning struct values.

1) Complete the function definition for a function ComputeLocation that returns a struct of type GPSPosition.

```
(double latitude, double
longitude) {

     ...

}
```

Check Show answer

2) Complete the function to return the calculated elapsed time, which gets stored in elapsedTime.

```
TimeEntry CalcElapsedTime(int
startSecs, int endSecs) {

     TimeEntry elapsedTime;

     ...

     elapsedTime.totalSecs =
endSecs - startSecs;

     elapsedTime.hours =
(endSecs - startSecs) / 3600;

     ...

     ;

}
```

Check Show answer

Likewise, a variable of a struct type can be a function parameter. And just like other types, a pass by value parameter would copy the item, while a pass by reference parameter would not.

PARTICIPATION ACTIVITY 7.2.4: Functions with struct parameters.

1) Complete the function definition for a function CalcSpeed that returns a double value and has two struct type

parameters startLoc and endLoc (in that order) of type GPSPosition.

```
double CalcSpeed(
                                  )
{
   ...
}
```

Check Show answer

2) Complete the following statement to calculate the speed between gpsPos1 and gpsPos2 by making a call to the CalcSpeed function.

```
double vehicleSpeed = 0.0;
GPSPosition gpsPos1;
GPSPosition gpsPos2;
...
vehicleSpeed =
                                  ;
...
```

Check Show answer

7.3 Structs and arrays

The power of structs becomes even more evident when used in conjunction with arrays. Consider a TV watching time program where a user can enter a country name, and the program outputs the daily TV watching hours average for a person in that country. One approach uses two same-sized arrays, one to hold names, and the other to hold numbers corresponding to each name. Instead of those two arrays, a struct allows for declaration of just one array that stores items that each have name and number data members.

Figure 7.3.1: An array of struct items rather than two arrays of more basic types.

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

const int MAX_COUNTRY_NAME_LENGTH = 50;

typedef struct CountryTvWatch_struct {
   char countryName[50];
   int tvMinutes;
} CountryTvWatch;

int main(void) {
   // Source: www.statista.com, 2010
   const int NUM_COUNTRIES = 4;

   CountryTvWatch countryList[NUM_COUNTRIES];
   char countryToFind[MAX_COUNTRY_NAME_LENGTH];
   bool countryFound = false;
   int i = 0;

   strcpy(countryList[0].countryName, "Brazil");
   countryList[0].tvMinutes = 222;
   strcpy(countryList[1].countryName, "India");
   countryList[1].tvMinutes = 119;
   strcpy(countryList[2].countryName, "U.K.");
   countryList[2].tvMinutes = 242;
   strcpy(countryList[3].countryName, "U.S.A.");
   countryList[3].tvMinutes = 283;

   printf("Enter country name: ");
   scanf("%s", countryToFind);

   countryFound = false;
   for (i = 0; i < NUM_COUNTRIES; ++i) { // Find country's index
      if (strcmp(countryList[i].countryName, countryToFind) == 0) {
         countryFound = true;
         printf("People in %s watch₩n", countryToFind);
         printf("%d minutes of TV daily.₩n", countryList[i].tvMinutes);
      }
   }
   if (!countryFound) {
      printf("Country not found, try again.₩n");
   }

   return 0;
}
```

```
Enter country name: U.S.A.
People in U.S.A. watch
283 minutes of TV daily.
...
Enter country name: UK
Country not found, try again.
...
Enter country name: U.K.
People in U.K. watch
242 minutes of TV daily.
```

Note that the countryList variable is declared as `CountryTvWatch countryList[NUM_COUNTRIES]`, meaning an array of items of type CountryTvWatch. Thus, each array element will have memory allocated for the struct's two data members, countryName and tvMinutes.

The notation `countryList[i].countryName` is equivalent to `(countryList[i]).countryName`, because the member access operator is evaluated left-to-right (as are any equal-precedence operators). The left-to-right member access operator evaluation is well-known among programmers so parentheses are typically omitted.

PARTICIPATION ACTIVITY 7.3.1: Using structs with arrays.

1) Declare the array countryList of 5

CountryTvWatch elements

Check Show answer

2) Given an array countryList consisting of 5 CountryTvWatch struct elements, write a statement that assigns the value of the 0th element's tvMinutes data member to the variable countryMin.

Check Show answer

3) Given an array countryList consisting of 5 CountryTvWatch struct elements, write one statement that copies element 4's struct values to element 0's.

Check Show answer

PARTICIPATION ACTIVITY 7.3.2: Modify the TV watch program.

Finish the PrintCountryNames() function to print all country names in the list.

Load default template...

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

const int MAX_COUNTRY_NAME_LENGTH = 50;

typedef struct CountryTvWatch_struct {
   char countryName[50];
   int tvMinutes;
} CountryTvWatch;

void PrintCountryNames(CountryTvWatch ctryList[], int numCo
{
   printf("FIXME: Finish PrintCountryNames()");
   return;
}

int main(void) {
   // Source: www.statista.com, 2010
```

```
USA
```

Run

7.4 Structs, arrays, and functions: A seat reservatior

A programmer commonly uses structs, arrays, and functions together. Consider a program that allows a reservations agent to reserve seats for people, useful for a theater, an airplane, etc. The below program defines a Seat struct whose data members are a person's first name, last name, and the amount paid for the seat. The program declares an array of 5 seats to represent the theater, airplane, etc., initializes all seats to being empty (indicated by a first name of "empty"), and then allows a user to enter commands to print all seats, reserve a seat, or quit.

Figure 7.4.1: A seat reservation system involving a struct, arrays, and functions.

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

typedef struct Seat_struct {
   char firstName[50];
   char lastName[50];
   int  amountPaid;
} Seat;

/*** Functions for Seat ***/

void SeatMakeEmpty(Seat* seat) {
   strcpy((*seat).firstName, "empty");
   strcpy((*seat).lastName,  "empty");
   (*seat).amountPaid = 0;

   return;
}

bool SeatIsEmpty(Seat seat) {
   return (strcmp(seat.firstName, "empty") == 0);
}

void SeatPrint(Seat seat) {
   printf("%s ", seat.firstName);
   printf("%s, ", seat.lastName);
   printf("Paid: %d₩n", seat.amountPaid);

   return;
}
/*** End functions for Seat ***/

/*** Functions for array of Seat ***/
void SeatsMakeEmpty(Seat seats[], int numSeats) {
   int i = 0;

   for (i = 0; i < numSeats; ++i) {
      SeatMakeEmpty(&seats[i]);
   }

   return;
}
```

```
Enter command (p/r/q): p
0: empty empty, Paid: 0
1: empty empty, Paid: 0
2: empty empty, Paid: 0
3: empty empty, Paid: 0
4: empty empty, Paid: 0

Enter command (p/r/q): r
Enter seat num: 2
Enter first name: John
Enter last name: Smith
Enter amount paid: 500
Completed.

Enter command (p/r/q): p
0: empty empty, Paid: 0
1: empty empty, Paid: 0
2: John Smith, Paid: 500
3: empty empty, Paid: 0
4: empty empty, Paid: 0

Enter command (p/r/q): r
Enter seat num: 2
Seat not empty.

Enter command (p/r/q): r
Enter seat num: 3
Enter first name: Mary
Enter last name: Jones
Enter amount paid: 198
Completed.

Enter command (p/r/q): p
0: empty empty, Paid: 0
1: empty empty, Paid: 0
2: John Smith, Paid: 500
3: Mary Jones, Paid: 198
4: empty empty, Paid: 0

Enter command (p/r/q): q
Quitting.
```

```
void SeatsPrint(Seat seats[], int numSeats) {
   int i = 0;

   for (i = 0; i < numSeats; ++i) {
      printf("%d: ", i);
      SeatPrint(seats[i]);
   }

   return;
}
/*** End functions for array of Seat ***/

int main(void) {
   const int NUM_SEATS = 5;
   char userKey = '-';
   int  seatNum = 0;
   Seat allSeats[NUM_SEATS];
   Seat currSeat;

   SeatsMakeEmpty(allSeats, NUM_SEATS);

   while (userKey != 'q') {
      printf("Enter command (p/r/q): ");
      scanf(" %c", &userKey);

      if (userKey == 'p') { // Print seats
         SeatsPrint(allSeats, NUM_SEATS);
         printf("\n");
      }
      else if (userKey == 'r') { // Reserve seat
         printf("Enter seat num: ");
         scanf("%d", &seatNum);

         if (!SeatIsEmpty(allSeats[seatNum])) {
            printf("Seat not empty.\n\n");
         }
         else {
            printf("Enter first name: ");
            scanf("%s", currSeat.firstName);
            printf("Enter last name: ");
            scanf("%s", currSeat.lastName);
            printf("Enter amount paid: ");
            scanf("%d", &currSeat.amountPaid);

            allSeats[seatNum] = currSeat;

            printf("Completed.\n\n");
         }
      }
      // FIXME: Add option to delete reservations
      else if (userKey == 'q') { // Quit
         printf("Quitting.\n");
      }
      else {
         printf("Invalid command.\n\n");
      }
   }

   return 0;
}
```

The programmer first defined several functions related to the Seat struct, such as checking if a seat is empty or printing a seat. The programmer then defined some functions related to an *array* of seat items. To distinguish, the programmer named the former starting with Seat and the latter starting with Seats.

The SeatMakeEmpty() function uses pass by pointer to update the information within an individual seat. Remember that to update a variable passed by pointer, the program must use the dereference operator * to access the value pointed to by the pointer. When the variable is a pointer to a structure, both the dereference operator and the member access operators must be used together. In this case, the member access operator has precedence, so parentheses are used to dereference the pointer first:

Construct 7.4.1: Dereferencing a pointer to a struct.

```
(*variableName).memberName
```

The programmer left a "FIXME" comment indicating that the program also requires the ability to delete a reservation. That functionality is straightforward to introduce, just requiring the user to enter a seat number and then making use of the existing SeatMakeEmpty() function.

Notice how main() is relatively clean, dealing mostly with the user commands, and then using functions to carry out the appropriate work. Actually, the "reserve seat" command could be improved; main() currently fills the reservation information (e.g., "Enter first name..."), but main() would be cleaner if it just called a function as `SeatFillReservationInfo(&currSeat)`.

The seat reservation program loses all its information when exited. An improvement is to save all reservation information in a file. Commands 's' and 'g' would save and get information to/from a file, respectively.

PARTICIPATION ACTIVITY 7.4.1: Seat reservation example with struct, array, and functions.

Refer to the above example.

1) The number of seats is 5.

○ True

○ False

2) SeatsMakeEmpty() has a loop that sets each seat in the seats array to have a first name of "empty".

○ True

○ False

3) SeatIsEmpty() checks if all the seats in the array are empty.

○

True

○ False

4) Deleting a reservation would reduce the array size from 5 down to 4.

○ True

○ False

PARTICIPATION ACTIVITY 7.4.2: Introduce delete behavior to the reservation program.

Modify main() to allow the user to enter command 'd', followed by the user entering a seat number. Call SeatMakeEmpty() to delete the seat.

Load default template...

```

#include <stdio.h>
#include <string.h>
#include <stdbool.h>

typedef struct Seat_struct {
   char firstName[50];
   char lastName[50];
   int  amountPaid;
} Seat;

/*** Functions for Seat ***/

void SeatMakeEmpty(Seat* seat) {
   strcpy((*seat).firstName, "empty");
   strcpy((*seat).lastName,  "empty");
   (*seat).amountPaid = 0;

   return;
}

```

```
p
r 2 John Smith 500
p
```

Run

7.5 Separate files for structs

Programmers typically put all code for a struct into two files, separate from other code.

Table 7.5.1: Typical two files per struct.

StructName.h	Contains the struct definition, including data members and related function

	declarations.
StructName.c	Contains related function definitions.

A file that uses the struct, such as a main file or StructName.c, must include StructName.h. The .h file's contents are sufficient to allow compilation, as long as the corresponding .c file is eventually compiled into the program too.

Figure 7.5.1: Using two separate files for a struct.

File: StoreItem.h

```
#ifndef STOREITEM_H
#define STOREITEM_H

typedef struct StoreItem_struct {
   int weightOunces;
   // (other fields omitted for brevity)
} StoreItem;

void StoreItemSetWeightOunces
        (StoreItem* storeItem, int
weightOunces);
void StoreItemPrint(StoreItem storeItem);

#endif
```

File: StoreItem.c

```
#include <stdio.h>
#include "StoreItem.h"

void StoreItemSetWeightOunces
        (StoreItem* storeItem, int weightOunces) {
   storeItem->weightOunces = weightOunces;
   return;
}

void StoreItemPrint(StoreItem storeItem) {
   printf("Weight (ounces): %d\n",
storeItem.weightOunces);
   return;
}
```

File: main.c

```
#include <stdio.h>
#include "StoreItem.h"

int main() {
   StoreItem item1;

   StoreItemSetWeightOunces(&item1, 16);
   StoreItemPrint(item1);

   return 0;
}
```

Compilation example

```
% gcc -Wall -Wextra  -std=c99 -pedantic StoreItem.c main.c
% a.out
Weight (ounces): 16
```

The figure shows how all the .c files might be listed when compiled into one program. Note that the .h file is *not* one of the listed files, as it is included in the appropriate .c files.

Sometimes multiple small related structs are grouped into a single file, to avoid a proliferation of files. But for typical structs, good practice is to create a unique .c and .h file for each struct.

For independent development and faster compilation, each struct file is typically compiled individually into an object file, and then later linked with a main file. Such compilation is discussed in another section on modular compilation.

PARTICIPATION ACTIVITY 7.5.1: Separate files.

1) Commonly a struct definition and associated function definitions are contained entirely in their own .h file.

- ○ True
- ○ False

2) The .c file for a struct should #include the associated .h file.

- ○ True
- ○ False

3) A drawback of the separate file approach is longer compilation times.

- ○ True
- ○ False