KU LEUVEN

C: an introduction

Structures - basics

User-Defined Types

- C provides facilities to define one's own types.
- These may be a composite of basic types (`int`, `double`, etc) and other user-defined types.
  - Array: homogeneous data
  - Structure: heterogeneous data
- The most common user-defined type is a structure, defined by the keyword `struct`.

Structures

- *A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling*
- They assist program organisation by
  - Grouping logically related data, and giving this set of variables a higher-level name and more abstract representation.
  - Enabling related variables to be manipulated as a single unit rather than as separate entities.
  - Reducing the number of parameters that need to be passed between functions.
  - Providing another means to return multiple values from a function.

Structure Syntax

- A structure is defined by the keyword **`struct`** followed by a set of variables enclosed in braces.
- Consider the following structure to represent a person's details.

```
struct Personnel {
    char name[100];
    int age;
    double height;
    };
```

- The variables **`name`**, **`age`** and **`height`** are called *members* of the structure type **`Personnel`** (a.k.a. *tag* - acting as a template)
- **Style note:** Structures can be given names with Capital first letters. Distinguish them from variables and functions (lowercase first letter), and symbolic constants (all uppercase).

Declaring Structure Variables

There are two ways to define variables of a particular structure type.

1. Declare them at the structure definition.

```
struct Personnel {
    char name[100];
    int age;
    double height;
} p1, p2, p3; /* Define 3 variables */
```

2. Define the variables at some point *after* the structure definition.

```
struct Personnel p1, p2, p3; /* Define 3 variables */
struct Personnel pa[3]; /* Define array of structure */
```


Typedef

- The keyword **`typedef`** provides a mechanism for creating new data type names.
  `typedef <data type definition> <data type name>;`
- It does not create new types, just new names (synonyms) for existing types.

```
typedef int Length;
Length len, maxlen;
Length lengths[50];
```

- **`typedef`** provides a simplification in structure declaration syntax.

```
typedef struct {
    int x;
    int y;
} Coord;
Coord p1, p2;
```

Initialising Structure Variables

- A structure may be initialised when it is defined using brace notation.

```
struct Personnel captain = {"Fred", 37, 1.83};
```

- The order of values in the initialiser list matches the order of declarations in the structure.

Accessing Members

- Members of a structure type may be accessed via the "." member operator (dot operator).

```
struct Personnel captain;

strcpy(captain.name, "Fred");
captain.age = 37;
captain.height = 1.83;

printf("%s is %d years old.",
captain.name, captain.age);
```

Nested Structures

- Structures may be defined inside other structures.

```
struct Payroll {
    struct Personnel person;
    double amount;
};
```

- To access lower-level members, need to use member operator multiple times.

```
struct Payroll lieutenant;
  lieutenant.person.height = 2.1;
  lieutenant.amount = 75.4;
```

KU LEUVEN

```c
/*
struct_point_xy.c
 */
#include <stdio.h>

struct  point
{
   int x;
   int y;
};

struct point x1;
struct point x2 = {200, 300};

void pointinfo(struct point xx);

int main()
{
   printf ("x1 without initialisation \n");
   pointinfo (x1);

   x1.x = 100;
   x1.y = 100;

   printf ("x1 after initialisation \n");
   pointinfo (x1);

   printf ("x2 after initialisation \n");
   pointinfo (x2);

   x1.x += x2.x;
   printf ("x1  summing x-dim of x2 \n");
   pointinfo (x1);

   return 0;
}

void pointinfo ( struct point xx )
{
    printf ("dim1 %d \t dim2 %d \n", xx.x, xx.y);
}
```

```
frankvp@CRD-L-08004:.../Structures$ gcc struct_point_xy.c -o struct_point_xy
frankvp@CRD-L-08004:.../Structures$ ./struct_point_xy
x1 without initialisation
dim1 0   dim2 0
x1 after initialisation
dim1 100         dim2 100
x2 after initialisation
dim1 200         dim2 300
x1  summing x-dim of x2
dim1 300         dim2 100
frankvp@CRD-L-08004:.../Structures$
```

KU LEUVEN

```c
/*
struct_manip_1.c
 manipulate the data contained in a strucure
 */
#include <stdio.h>
#include <string.h>

typedef struct record
   { char name[20]; int age; float reward;} Person;

void display(char *name, int age, float reward);
void raise(Person * a);

int main ()
   {
   Person p1, p2;

   strcpy(p1.name, "Joe Brown");
   p1.age = 21;
   p1.reward = 123.4;
   display (p1.name, p1.age, p1.reward);
   raise(&p1);
   display (p1.name, p1.age, p1.reward);

   return 0;
   }


void display(char *name, int age, float reward)
   {
   printf("name is %s \nage is %d \nreward is %lf \n", name, age, reward);
   }

void raise(Person * a)
   {
   a->reward = a->reward * 1.15;
   }
```

```
frankvp@CRD-L-08004:.../Structures$ gcc struct_manip_1.c -o struct_manip_1
frankvp@CRD-L-08004:.../Structures$ ./struct_manip_1
name is Joe Brown
age is 21
reward is 123.400002
name is Joe Brown
age is 21
reward is 141.910004
frankvp@CRD-L-08004:.../Structures$
```

KU LEUVEN

Operations on Structures

- can be copied, assigned

```
struct Personnel p1 = {"Fred", 37, 1.83};
struct Personnel p2;

    p2 = p1;        /* Valid. */

    if (p1 == p2)   /* Invalid. Won't compile. */
        printf("People are equal\n");

    if (strcmp(p1.name, p2.name) == 0 && /* Valid. */
            p1.age == p2.age &&
            p1.height == p2.height)
                printf("People are equal\n");
```

- can be passed as an argument in a function
- can be returned as a result from a function
- can not be compared to each other

KU LEUVEN