FIT ICT Software Development

Lecture 10

Building Structures

Lecturer : Charles Tyner

FIT ICT Software Development

Structures

Can contain one or more variables

Variables can be the same or different types

Grouped together in custom-designed structure

Referenced via structure name

Helps in organising complex data

Mimics a database

Declared using **struct** keyword, followed by curly brackets (braces) containing the variable members and terminated by a semi-colon

FIT ICT Software Development

Grouping in a Structure

Structure containing members describing x and y coordinates of a point on a graph:

```
struct cords
{
  int x;
  int y;
}point;
```

point is an optional tag-name

Included between closing brace and terminating semi-colon

Multiple tags can be included in a comma-separated list

Each ***struct*** member can be referenced using the "." dot operator in the form ***point.x***

FIT ICT Software Development

Grouping in a Structure

```
#include <stdio.h>

struct coords
{
 int x ;
 int y ;
}point ;
struct coords top ;

int main()
{
 point.x = 5 ;  point.y = 8 ;
 top.x = 15 ;  top.y = 24 ;

 printf("\npoint x: %d, point y: %d\n", point .x, point .y ) ;
 printf("\ntop x: %d, top y: %d\n", top .x, top .y ) ;
 return 0;
}
```

FIT ICT Software Development

Grouping in a Structure

After header file ***stdio.h*** is included, structure is declared. Immediately, another instance of the ***struct*** is created

Main function opens with initialisation of both members of both structures

Values of members of structures then displayed using *printf()* function and a zero returned

Save, compile, execute and study

FIT ICT Software Development

Defining Type Structures

Using ***typedef*** keyword before the ***struct*** keyword creates a custom datatype

Can be used as a prototype to declare other structures

Reduces use of ***struct*** keyword and can make program clearer and easier to read

Tag names usually capitalised to aid recognition

Structures can be nested, but individual members must be referenced using two ***dot*** operators

FIT ICT Software Development

Defining Type Structures

```
#include <stdio.h>

typedef struct
{
 int x ;
 int y ;
}Point ;

Point top = { 15, 24 } ;
Point btm ;

typedef struct
{
 Point a ;
 Point b ;
}Box ;
Box rect = { 6, 12, 30, 20 };
```

FIT ICT Software Development

Defining Type Structures

```
int main()
{
 btm.x = 5 ;
 btm.y = 8 ;
 printf("\nTop x: %d, y: %d\n", top.x, top.y ) ;
 printf("Bottom x: %d, y: %d\n", btm.x, btm.y ) ;
 printf("\nPoint a x: %d", rect.a.x ) ;
 printf("\nPoint a y: %d", rect.a.y ) ;
 printf("\nPoint b x: %d", rect.b.x ) ;
 printf("\nPoint b y: %d\n", rect.b.y ) ;

 return 0;
}
```

FIT ICT Software Development

Grouping in a Structure

After header file ***stdio.h*** is included, un-named structure is declared defining a data type with two members and a tag.

Immediately, two variables of the structure-defined data type are created, with the members of one of them being initialised

A nested type definition is created by declaring a new data type, the members of which are instances of the first data type. This is also given a tag name

A variable of this latest data type is created and initialised with a comma-delimited list of values

FIT ICT Software Development

Grouping in a Structure

A main function is added which starts with the initialisation of the uninitialized variable, an alternative method of initialisation.

Various values are output showing how to reference members of structures

A zero is returned as required

Save, compile, execute and study

FIT ICT Software Development

A Word About Pointers

Pointers are very useful in C.

They are variables that store memory address of other variables

Declared in same way as other variables, but name is preceded by an asterisk "*"

Pointer must be of same data type as variable it references

Assigned address of variable by use of & addressof operator

Pointer references address it stores.

Dereferenced by preceding pointer name with asterisk, and then references value

Can store new value at address, thereby assigning new value to regular variable

FIT ICT Software Development

Pointers

```
#include <stdio.h>

int main()
{
 int num = 8 ;
 int *ptr = &num ;
 printf("Regular variable contains: %d\n", num ) ;
 printf("Pointer variable contains: 0x%p\n", ptr ) ;
 printf("Pointer points to value: %d\n\n", *ptr ) ;

*ptr = 12 ;
 printf("Regular variable contains: %d\n", num ) ;
 printf("Pointer variable contains: 0x%p\n", ptr ) ;
 printf("Pointer points to value: %d\n\n", *ptr ) ;

 return 0;
}
```

FIT ICT Software Development

Pointing to Structures

Pointers to ***struct*** data types created the same way.

Pointer hold address of beginning of region of memory used to store member data

Special arrow operator **->** used instead of dot operator to reference members

Pointers a powerful and much-used feature of C

This slide, and listing that follows, show only a single usage.

Full exploration beyond scope of course!

FIT ICT Software Development

Pointing to Structures

```
#include <stdio.h>
typedef struct
{
  char *name ; char *pop ;
} County ;

int main()
{
 County dn, ck, gy, *ptr ;
 dn.name = "Dublin" ;
 dn.pop = "1,270,603" ;
 printf("\n%s, Population: %s\n", dn.name, dn.pop) ;
```

FIT ICT Software Development

Pointing to Structures

```
 ptr = &ck ;
 ptr->name = "Cork" ;
 ptr->pop = "518,128" ;
printf("\n%s, Population: %s\n", ck.name, ck.pop) ;

 ptr = &gy ;
 ptr->name = "Galway" ;
 ptr->pop = "250,541" ;
 printf ("\n%s, Population: %s\n", ptr.name, ptr.pop) ;

 return 0;
}
```

FIT ICT Software Development

Pointing to Structures

After including library header file, a structure is prototyped with two members and a tag name

In main function, data type is used in declaration of three variables and a pointer variable

Values are assigned to members of first variable using dot operator

Stored values displayed using dot operator

Access to values is directly via the structure name

FIT ICT Software Development

Pointing to Structures

Address of second variable is assigned to pointer variable

Members of second variable initialised via arrow operator and retrieved and displayed using the dot operator

Address of third variable is assigned to pointer variable

Members of third variable initialised, retrieved and displayed using the arrow operator

FIT ICT Software Development

Passing Structures to Functions

- Structure members can be stored in an array, like any data type
- Array declared as normal, but values passed in a slightly different way
- Each comma-separated list of member values enclosed within curly brackets
- Structures can also be passed as arguments to functions
- Structures data type must be specified in prototype declaration and in definition

FIT ICT Software Development

Passing Structures to Functions

- Structure members can be stored in an array, like any data type
- Array declared as normal, but values passed in a slightly different way
- Each comma-separated list of member values enclosed within curly brackets
- Structures can also be passed as arguments to functions
- Structures data type must be specified in prototype declaration and in definition

FIT ICT Software Development

Passing Structures to Functions

```
#include <stdio.h>

typedef struct
{
  char *name ;
  int quantity ;
} Item ;

Item fruits[3] = { { "Apple", 10 } , { "Orange", 20 } ,{ "Pear", 30 } };

void display( Item val, Item *ref) ;

void display( Item val, Item *ref)
 { printf("%s: %d\n", val.name, val.quantity) ;
   val.name = "Banana" ; val.quantity = 40 ;
   printf("%s: %d\n", val.name, val.quantity) ;
```

FIT ICT Software Development

Passing Structures to Functions

```
  printf("%s: %d\n", fruits[0].name, fruits[0].quantity) ;
  ref->name = "Peach" ;
  ref->quantity = 50 ;
  printf("%s: %d\n", fruits[0].name, fruits[0].quantity) ;
}

int main()
{
  display(fruits[0], &fruits[0] ) ;

 return 0;
}
```

FIT ICT Software Development

Passing Structures to Functions

Standard I/O header file included

Un-named structure declared defining data type with two members and a tag name

Next, an array of structure's data type is declared, initialising each member of three structures

Function prototype added to pass a structure variable and a pointer variable to a structure

Now, function is defined. First it displays values of passed arguments

FIT ICT Software Development

Passing Structures to Functions

Function block then changes values passed and displays the new values

Printing the values by direct reference to the array shows that original values have not changed.

Using a regular variable to pass values means function operates on a copy and change occurs only locally

Next, original values are changed and output

Finally, main function simply calls the display function, passing index zero.