CSC180: Lecture 28

Wael Aboulsaadat

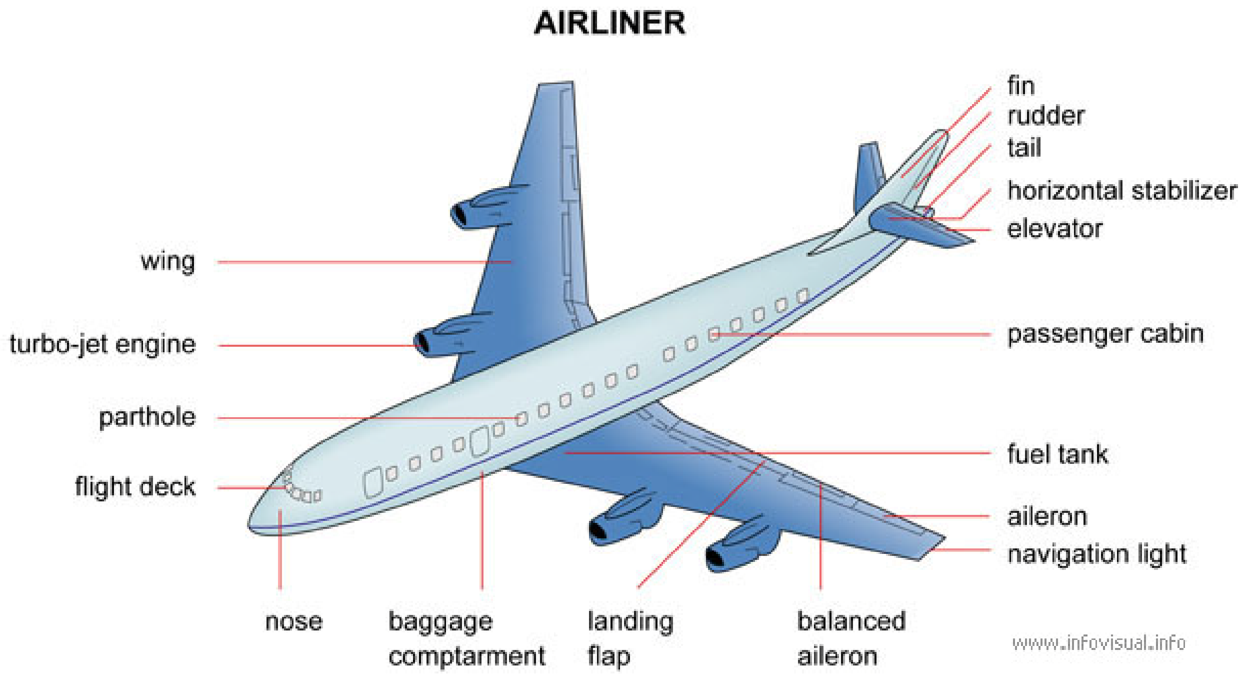
wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

Advantages of Hierarchical Structures

- Capture relations between real-life entities
- Encapsulation of related variables
- E.g.



Structure Pointers

- Calculate average score

```
double average(struct student *p, int n)
{
   int i, total =0;
   for (i = 0; i < n ; p++, i++) {
      total = total + p->score;
   }
   return (total/(double)n);
}
```

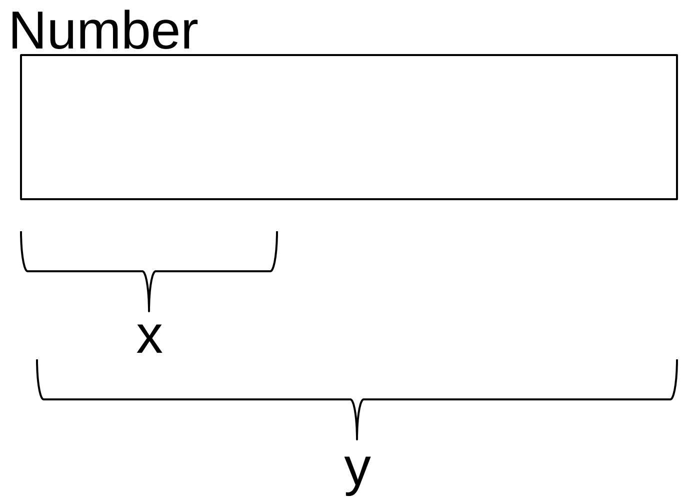
```
int main()
{
   struct student  cs180[] = {
      “John”, 10,
      “Adam”, 13,
      “Lee”, 17
   };
   average(cs180,
      sizeof(cs180)/sizeof(struct student));
   return 0;
}
```

Unions

- `union`
  - Memory that contains a variety of objects over time
  - Only contains one data member at a time
  - Members of a `union` share space
  - Conserves storage
  - Only the last data member defined can be accessed

- `union` definitions
  - Similar to struct definitions

```
union Number {
  int x;
  float y;
};
union Number value;
```



Unions Operations

- Valid `union` operations
  - Assignment to `union` of same type: `=`
  - Taking address: `&`
  - Accessing union members: `.`
  - Accessing union members using pointers: `->`

```c
/* Fig. 10.5: fig10_05.c
   An example of a union */
#include <stdio.h>

/* number union definition */
union number {
   int x;
   double y;
}; /* end union number */

int main( void )
{
   union number value; /* define union variable */

   value.x = 100; /* put an integer into the union */
   printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n\n\n",
      "Put a value in the integer member",
      "and print both members.",
      "int:", value.x,
      "double:", value.y );

```

Union definition

Union definition must end with semicolon

Note that **y** has no value

```c
   value.y = 100.0; /* put a double into the same union */
   printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n",
      "Put a value in the floating member",
      "and print both members.",
      "int:", value.x,
      "double:", value.y );

   return 0; /* indicates successful termination */

} /* end main */
```

Giving **y** a value removes **x**’s value

```
Put a value in the integer member
and print both members.
int:
  100

double:
  -92559592117433136000000000000000000000000000000000000000000000.000000


Put a value in the floating member
and print both members.
int:
  0

double:
  100.000000
```

Structs with Union

- /* The program on the next 3 slides creates a union and makes it a member of <u>struct personal</u> which is, in turn, a member of <u>struct identity</u>. The union uses the same memory location for either rank or a character string (deg) depending on the answer to the prompt for student status in main( ) */

Structs with Union (cont.)

```
#include <stdio.h>

union status
{
  int rank ;
  char deg[4] ;
} ;
```

```
struct personal
{
  long id ; float gpa ;
  union status level ;
} ;

struct identity
{
  char name[30] ;
  struct personal student ;
} ;
```

Structs with Union (cont.)

```
int main( )
{
   struct identity jb = {"Joe Brown"},
                       *ptr = &jb;
   char u_g;

   jb.student.id = 123456789 ;
   jb.student.gpa = 3.4 ;

printf ("Enter student status - u or g\n");

   scanf ("%c", &u_g);
   if (u_g == 'u')
     { printf ("Enter rank -- 1, 2, 3, 4 or 5\n");
       scanf ("%d", &jb.student.level.rank);
       printf ("%s is level %d\n" , jb.name ,
                  jb.student.level.rank);
     } /* End of if statement */
```

Structs with Union (cont.)

```
else
 {
    printf ("Enter degree sought -- ms or phd\n");
    scanf ("%s", &jb.student.level.deg);
    printf ("%s is a %s candidate\n”, jb.name , jb.student.level.deg );
 }   /* End of else statement */

 printf ("%s %ld %f\n” , jb.name , jb.student.id , jb.student.gpa );
 printf ("%s%ld %f\n” , ptr->name , ptr->student.id , ptr->student.gpa );

}  /*  End of program */
```

User Defined Data Types (typedef)

- The C language provides a facility called *typedef* for creating synonyms for previously defined data type names. For example, the declaration:

```
typedef  int  Length;
```

makes the name *Length* a synonym (or alias) for the data type *int*.

- The data “type” name *Length* can now be used in declarations in exactly the same way that the data type *int* can be used:

```
Length   a, b, len ;
Length   numbers[10] ;
```

Typedef & Struct

- Often, *typedef* is used in combination with *struct* to declare a synonym (or an alias) for a structure:

```
typedef struct                    /* Define a structure */
{
   int label ;
   char letter;
   char name[20] ;
} Some_name         /* The "alias" is Some_name */

Some_name  mystruct ;    /* Create a struct variable */
```