Chapter - 12 Advanced Types

Structures

Arrays allow you to create a data collection for a single type:

```
    int data[100];    // Collection of 100 integers
```

Structures allow you to collect data of different types:

```
    struct bin {
        char    name[30];   // name of the part
        int     quantity;   // how many are in the bin
        int     cost;       // The cost of a single part
                            //  (in cents)
    } printer_cable_box;    // where we put the
                            // print cables
```

The general form of a structure definition is:

```
 struct structure-name {
     field-type field-name  // comment
     field-type field-name  // comment
    . . . .
 } variable-name;
```

Structure Usage

```
// Place for terminal cables
struct bin terminal_cable_box;
```

The *structure-name* part of the definition may be omitted.

```
struct {
    char    name[30];   // name of the part
    int     quantity;   // how many are in the bin
    int     cost;       // The cost of a
                        // single part  (in cents)
} printer_cable_box;    // where we put the
                        // print cables
```

The *variable-name* may also be omitted. This would define a structure type, but no variables.

```
struct bin {
    char    name[30];   // name of the part
    int     quantity;   // how many are in the bin
    int     cost;       // The cost of a
                        // single part  (in cents)
};
```

Usage

Elements in a structure (called fields) are accessed by:

```
    variable.field
```

Example:

```
    // $12.95 is the new price
    printer_cable_box.cost = 1295;
```

Initialization

```
/*
 * Printer cables
 */
struct bin {
    char    name[30];   // name of the part
    int     quantity;   // how many are in the bin
    int     cost;       // Single part cost (in cents)
};
struct bin printer_cable_box = {
    "Printer Cables",   // Name of the item in the bin
    0,                  // Start with empty box
    1295                // cost -- $12.95
};
```

One step initialization:

```
struct bin {
    char    name[30];   // name of the part
    int     quantity;   // how many are in the bin
    int     cost;       // Single part cost (in cents)
} printer_cable_box = {
    "Printer Cables",   // Name of the item in the bin
    0,                  // Start with empty box
    1295                // cost -- $12.95
};
```

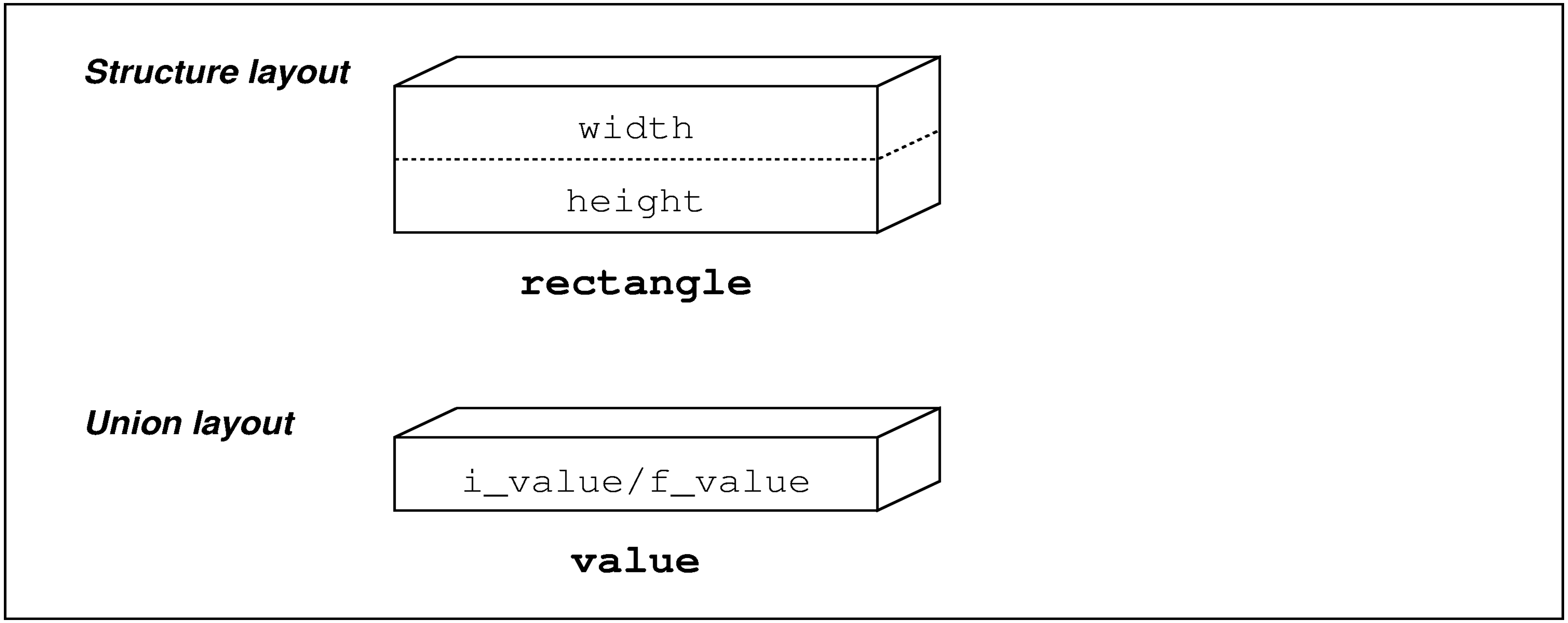
Unions

Structure -- each field is stored in a different location. Fields do not interfere with each other.

Union -- each field is stored in the same location. Changing one field puts garbage in the other fields.

```
union value {
    long int i_value; // long int version of value
    float f_value;    // floating version of value
}
```

Union Layout



Union Usage

```
/*
 * Define a variable to hold an integer or
 * a real number (but not both)
 */
union value {
    long int i_value; // The real number
    float f_value;    // The floating point number
} data;

int i;              // Random integer
float f;            // Random floating point number
```

Union Usage

```
int main(){
    data.f_value = 5.0;
    data.i_value = 3;   // data.f_value overwritten

    i = data.i_value;   // legal

    f = data.f_value;   // not legal, generates
                        // unexpected results

    data.f_value = 5.5; // store in f_value
                        //clobber i_value

    i = data.i_value;   // not legal, generates
                        // unexpected results
```

Union Example

```
struct circle {
    int radius;   // Radius of the circle in pixels
};
struct rectangle {
    int height, width; // Size of the rect.(pixels)
}
struct triangle {
    int base;  // Length of in pixels
    int height;// Height of the triangle in pixels
};
```

Union Example

```
const int SHAPE_CIRCLE    = 0;    // Shape's circle
const int SHAPE_RECTANGLE = 1;    // Shape's rect.
const int SHAPE_TRIANGLE  = 2;    // Shape's tri.

struct shape {
    int kind;          // What kind of shape
    union shape_union {// Union to hold shape info.
        struct circle    circle_data;
        struct rectangle rectangle_data;
        struct triangle  triangle_data;
    } data;
};
```

typedef

General form:

```
    typedef type-declaration.
```

The type-declaration is the same as a variable declaration except a type name is used instead of a variable name.

Example:

```
   // Define the type “width” of an object
   typedef int width;
```

We can now use our new type:

```
    width box_width;
```

Enum Type

Poor coding:

```
typedef int day_of_the_week;// define type for week days


const int SUNDAY    = 0;
const int MONDAY    = 1;
const int TUESDAY   = 2;
const int WEDNESDAY = 3;
const int THURSDAY  = 4;
const int FRIDAY    = 5;
const int SATURDAY  = 6;


/* now to use it */
day_of_the_week today = TUESDAY;
```

Better coding:

```
enum day_of_the_week {SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
                        THURSDAY, FRIDAY, SATURDAY};


/* now use it */
enum day_of_the_week today = TUESDAY;
```