

module4

gar

Outline

Structures and File Management

Structure

- ▶ Used to group data values which are heterogeneous, and the data are related an entity
 - ▶ Array groups homogeneous data
- ▶ Structure is declared using `struct` data type
- ▶ E.g. To store a student name, roll number and branch, a structure can be defined as:

```
struct student {  
    char name[20];  
    int rollno;  
    char branch[20];  
};
```

- ▶ `name`, `rollno` and `branch` are called the *members* of the structure, which can be of any data type

General Form of struct definition

```
struct struct_tag {  
    type_1 var_1;  
    type_2 var_2;  
    .  
    .  
    type_n var_n;  
} structvar1, structvar2;
```

Accessing members of a structure

- ▶ A dot operator is used to access the members
- ▶ E.g. consider a struct point used to store x and y coordinates of a point:

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

- ▶ Declare and initialize a variable p

```
struct point p = {1,2};
```

- ▶ To print value x of p

```
printf("%d\n", p.x);
```

- ▶ To set a new value of x

```
p.x = 5;
```

Questions

1. Define a structure `book`, which must store the title of the book, author name, year of publication and the publisher's name
2. Define a structure `rectangle`, which stores two point structures (nested structure), to indicate the bottom left and top right points of a rectangle

Nested Structure

- ▶ Defining a rectangle structure (initializing and accessing the values)

```
struct rectangle {  
    struct point p1; /* bottom-left */  
    struct point p2; /* top-right */  
};  
  
int main()  
{  
    struct rectangle r = {{1,2},{5,5}};  
    printf("%d %d\n", r.p1.x, r.p1.y);  
    printf("%d %d\n", r.p2.x, r.p2.y);  
}
```

Array of Structures

- ▶ Array of struct data type is also possible:

- ▶ E.g.

```
struct point p[10];
```

declares an array of 10 points, and reading of the data can be done using dot operator

```
for (i=0; i<10; i++)  
    scanf("%d%d", &p[i].x, &p[i].y);
```


Structure and Function

- ▶ When passing a structure to a function, it's passed by value instead of reference
- ▶ Changing values in the called function will not change the structure variables in the calling function
- ▶ E.g.

```
void set(struct point p, int x, int y)
{
    p.x = x;
    p.y = y;
}
main()
{
    struct point a;
    set(a, x, y);
}
```

will not set the values of a, since a copy of the structure is sent instead of the address

Structure and Function

- ▶ To set the values of a structure, pass by reference and use `->` operator

```
void set(struct point *p, int x, int y)
{
    p->x = x; /* (*p).x = x; also is fine */
    p->y = y;
}

main()
{
    struct point a;
    set(&a, x, y);
}
```

Sending Array of Structures as Parameter

```
struct candidate {
    char name[50];
    int age;
};

void read_data(struct candidate c[], int n)
{
    int i;
    for (i=0; i<n; i++) {
        scanf("%s", c[i].name);
        scanf("%d", &c[i].age);
    }
}

int main ()
{
    struct candidate cand[10];
    read_data(cand, 2);
    return 0;
}
```

Type Definitions

- ▶ C provides a keyword `typedef` for creating new data type names
- ▶ E.g. If we are going to use `short int` to store dimensions of rectangles, `typedef` can be used

```
typedef short int Length;
```

- ▶ Then we may use the data type `Length` instead of `short int`
`Length h, w; /* represents height and width */`
- ▶ It depicts the intentions of the variables more clearly

Type Definitions

- ▶ As another example, a structure can be typedef in two ways

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

- ▶ or

```
typedef struct point {  
    int x;  
    int y;  
} Point; /* In this case, point tag is optional */
```

- ▶ We may use the new data type to declare point structures

```
main()  
{  
    Point p = {1,5};  
    ...  
}
```

File Management

- ▶ A file is a collection of data stored on disk
- ▶ C provides with several functions to work with files
- ▶ A file must be opened by using the function `fopen`, which returns a pointer to the opened file
 - ▶ it's called a *file pointer*

Opening a File

- ▶ General form of a call to `fopen` is

```
filepointer = fopen(filename, mode);
```

- ▶ Example usage, to open a file `names.txt` in read-only mode:

```
/* Opened in read-mode, other modes: "w", "a" */  
FILE *fp = fopen("names.txt", "r");  
if (fp == NULL)  
    printf("Error opening the file\n");
```

- ▶ If the file is not found, or necessary permissions are not there for the program, a `NULL` pointer will be returned
- ▶ Otherwise, a pointer to that file will be returned
- ▶ “w” opens a file to write. If the file already exists, it will be overwritten, otherwise it will be created
- ▶ “a” opens a file to append. If the file already exists, it will append output to end of the file, otherwise it will be created

Output using fprintf

- ▶ General form:

```
fprintf(destfile, "format string", list of variables);
```

- ▶ Sending output to a file

```
FILE *fo;  
int num = 119;  
  
fo = fopen("nums.txt", "w");  
if (fo != NULL)  
    fprintf(fo, "%d\n", num);
```


Input using fscanf

- ▶ General form

```
fscanf(sourcefile, "format string", list of variables);
```

- ▶ Receiving input from a file

```
FILE *fi;
```

```
int num;
```

```
fi = fopen("nums.txt", "r");
```

```
if (fi != NULL)
```

```
    fscanf(fi, "%d", &num);
```

- ▶ This reads a number from the file `nums.txt` and stores in the variable `num`

stdin and stdout

- ▶ `stdin` is called an input stream (standard input) and `stdout` is the output stream (standard output)

- ▶ The two statements below mean the same

```
fscanf(stdin, "%d", &num);  
scanf("%d", &num);
```

- ▶ Similarly, the two statements below mean the same

```
fprintf(stdout, "%d", num);  
printf("%d", num);
```

- ▶ Every program is provided with these pointers by default
- ▶ Another stream is available, which is called `stderr`, mainly used for redirecting error and diagnostic messages

Closing files

- ▶ After a file is used, it must be closed
- ▶ It's general form is

```
fclose(filepointer);
```

String input from a file

```
struptr = fgets(inputarea, n, source);
```

- ▶ Example

```
char inarea[15];
```

```
char *instring;
```

```
instring = fgets(inarea, 15, stdin);
```

- ▶ fgets reads at most 14 characters, then appends one NULL character

- ▶ If the entered string is : "Hello, world string"
- ▶ The string stored will be

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|--|---|---|---|---|---|--|---|----|
| H | e | l | l | o | , | | w | o | r | l | d | | s | \0 |
|---|---|---|---|---|---|--|---|---|---|---|---|--|---|----|

String output to a file

- ▶ General form

```
fputs(string, destfile);
```

- ▶ Example: echoing back whatever is typed

```
char inarea[20];
```

```
char *instring;
```

```
instring = fgets(inarea, 20, stdin);
```

```
while (instring != NULL) {
```

```
    fputs(instring, stdout);
```

```
    instring = fgets(inarea, 20, stdin);
```

```
}
```

File I/O for characters: `fgetc`

- ▶ General form of `fgetc`

```
ch = fgetc(source);
```

- ▶ Reads character from a file pointed by `source`
- ▶ Returns character converted to `int`, or `EOF` at the end of file

File I/O for characters: fputc

- ▶ General form of fputc

```
fputc(ch, destfile);
```

- ▶ `ch` is a character, and `destfile` is a pointer to a file or `stdout`

Example Programs

1. Write a program to replace all the letter 'a' with letter 'A', where the text is read from "indata.txt" and the output is stored in "outdata.txt"
2. Write a program to count the number of full stops and commas in the file named "howmany.txt", and display the result in the terminal

Program 1

```
#include <stdio.h>

int main()
{
    FILE *fi = fopen("indata.txt", "r");
    FILE *fo = fopen("outdata.txt", "w");
    int ch;
    if (fo != NULL) {
        ch = fgetc(fi);
        while (ch != EOF) {
            if (ch == 'a') {
                fputc('A', fo);
            } else {
                fputc(ch, fo);
            }
            ch = fgetc(fi);
        }
    }
    fclose(fi);
    fclose(fo);
    return 0;
}
```

Program 2

```
#include <stdio.h>

int main()
{
    FILE *fi;
    int comma = 0, dots = 0, ch;
    fi = fopen("howmany.txt", "r");
    if (fi != NULL) {
        ch = fgetc(fi);
        while (ch != EOF) {
            if (ch == '.')
                dots++;
            if (ch == ',')
                comma++;
            ch = fgetc(fi);
        }
        fclose(fi);
        printf("commas: %d, full stops: %d\n", comma, dots);
        return 0;
    }
```