COMP133 –COMPUTER AND PROGRAMMING

Files

Dr. Radi Jarrar
Department of Computer Science
Birzeit University

BIRZEIT UNIVERSITY

* The material in this lecture notes is adapted from the slides of Mr. Abdallah Karakra, Birzeit University, 2016

Binary Files

- Files containing binary numbers that are the computer's internal representation of each file component.

Binary Files

- Why do we need Binary files?
  - Smaller in size in comparison to text file
  - We can store entire data types and then we can re-read them again
  - We can store entire arrays and/or structs into binary files

Disadvantages of Binary Files

- A binary file created on one computer is not always readable on another type of computers.
- A binary file can not be created or modified in a word processor.
- To read a binary file: `fread`
- To write a binary file: `fwrite`

Opening a Binary File

- Add “b” to the **fopen** mode string
  - “rb” : read a binary file
  - “wb” : write a binary file
  - “ab” : append to a binary file

- `FILE *fp= fopen("myfile.bin", "rb");//read`
- `FILE *fp= fopen("myfile.bin","wb");//write`

Writing to a Binary File

- `size_tfwrite(const void * ptr, size_tsize, size_tcount, FILE * stream)`

- INPUT
- *A ptr to an array of elements (or just one)*
- The size of each element
- The number of elements
- Pointer to a FILE object that specifies an output stream (File pointer)

- OUTPUT
- Returns the number of elements written
- If return value is different than count, there was an error

Writing to a Binary File

```
FILE *fp = fopen("myfile.bin","wb");
...

int nums[] = {1,2,3};
fwrite(nums, sizeof(int), 3, fp);

double dub = 3.1;
fwrite(&dub, sizeof(double), 1, fp);
```

- **`sizeof`** operator finds the number of bytes used to store a data type

Reading a Binary File

- `size_t fread( void * ptr, size_tsize, size_tcount, FILE * stream )`

- INPUT
- A ptr to some memory of size at least (size * count)
- The size of each element to read
- The number of elements to read
- Pointer to a FILE object that specifies an input stream (File Pointer)

- OUTPUT
- Returns number of elements read
- If return value is different than count, there was an error or the end of the file was reached

Reading a Binary File

```
FILE *fp = fopen("myfile.bin","rb");
...
int nr;

int nums[3];
nr = fread(nums, sizeof(int), 3, fp);
//Check for errors

double dub;
nr = fread(&dub, sizeof(double), 1, fp);
//Check for errors
```

- **`sizeof`** operator finds the number of bytes used to store a data type

Example: Creating a Binary File of Integers

```
1.  FILE *binaryp;
2.  int   i;
3.
4.  binaryp = fopen("nums.bin", "wb");
5.
6.  for  (i = 2;   i <= 500;   i += 2)
7.      fwrite(&i, sizeof (int), 1, binaryp);
8.
9.  fclose(binaryp);
```

Example: Writing to a Binary File

```
#include <stdio.h>
#define SIZE 100
int main()
{
    int x=20,A[SIZE]={0,1,2,3};
    FILE* fptr_out=fopen("out.bin", "wb");

    fwrite(&x, sizeof(int),1,fptr_out);
    fwrite(A, sizeof(int),SIZE,fptr_out);
    fclose(fptr_out);
    return 0;
}
```

Example: Reading from a binary file

```
#include <stdio.h>
#define SIZE 100
int main()
{
    int x,A[SIZE];
    FILE* fptr_inp=fopen("in.bin", "rb");

    fread(&x, sizeof(int),1,fptr_inp);
    fread(A, sizeof(int),SIZE,fptr_inp);
    fclose(fptr_inp);
    return 0;
}
```

Example: writing and reading a complex number

```c
#include <stdio.h>
typedef struct {
 int real;
 int imag;

} complex_t;
int main()
{
    complex_t x1={2,3};
    complex_t x2={4,5};
    complex_t x3;
    FILE *fptr_inp;
    FILE *fptr_out;
    fptr_out=fopen("out.bin","wb");//for writing
    fwrite(&x1,sizeof(complex_t),1,fptr_out);
    fclose(fptr_out);
    fptr_inp=fopen("out.bin","rb");//for reading
    fread(&x3,sizeof(complex_t),1,fptr_out);
    fclose(fptr_inp);
    printf("%d %d\n%d %d",x3.real,x3.imag,x2.real,x2.imag);
    return 0;
}
```

Text File vs. Binary File

TABLE 11.5

Example	Text File I/O	Binary File I/O	Purpose

5	`for (i = 0; i < MAX; ++i)`
	`  fscanf(doub_txt_inp,`
	`  "%lf", &nums[i]);`
		`fread(nums, sizeof (double),`
		`  MAX, doub_bin_inp);`
			Fill array **nums** with type **double** values from input file.

6	`for (i = 0; i < MAX; ++i)`
	`  fprintf(doub_txt_outp,`
	`  "%e\n", nums[i]);`
		`fwrite(nums, sizeof (double),`
		`  MAX, doub_bin_outp);`
			Write contents of array **nums** to output file.

7	`n = 0;`
	`for (status =`
	`  fscanf(doub_txt_inp,`
	`  "%lf", &data);`
	`  status != EOF &&`
	`  n < MAX;`
	`  status =`
	`  fscanf(doub_txt_inp,`
	`  "%lf", &data))`
	`  nums[n++] = data;`
		`n = fread(nums,`
		`  sizeof (double),`
		`  MAX, doub_bin_inp);`
			Fill **nums** with data until **EOF** encountered, setting **n** to the number of values stored.

8	`fclose(plan_txt_inp);`
	`fclose(plan_txt_outp);`
	`fclose(doub_txt_inp);`
	`fclose(doub_txt_outp);`
		`fclose(plan_bin_inp);`
		`fclose(plan_bin_outp);`
		`fclose(doub_bin_inp);`
		`fclose(doub_bin_outp);`
			Close all input and output files.

Text File vs. Binary File

TABLE 11.5

Example	Text File I/O	Binary File I/O	Purpose

5	`for (i = 0; i < MAX; ++i)`
	`  fscanf(doub_txt_inp,`
	`  "%lf", &nums[i]);`
		`fread(nums, sizeof (double),`
		`  MAX, doub_bin_inp);`
			Fill array **nums** with type **double** values from input file.

6	`for (i = 0; i < MAX; ++i)`
	`  fprintf(doub_txt_outp,`
	`  "%e\n", nums[i]);`
		`fwrite(nums, sizeof (double),`
		`  MAX, doub_bin_outp);`
			Write contents of array **nums** to output file.

7	`n = 0;`
	`for (status =`
	`  fscanf(doub_txt_inp,`
	`  "%lf", &data);`
	`  status != EOF &&`
	`  n < MAX;`
	`  status =`
	`  fscanf(doub_txt_inp,`
	`  "%lf", &data))`
	`  nums[n++] = data;`
		`n = fread(nums,`
		`  sizeof (double),`
		`  MAX, doub_bin_inp);`
			Fill **nums** with data until **EOF** encountered, setting **n** to the number of values stored.

8	`fclose(plan_txt_inp);`
	`fclose(plan_txt_outp);`
	`fclose(doub_txt_inp);`
	`fclose(doub_txt_outp);`
		`fclose(plan_bin_inp);`
		`fclose(plan_bin_outp);`
		`fclose(doub_bin_inp);`
		`fclose(doub_bin_outp);`
			Close all input and output files.