Structured Programming

- File I/O

Dr Donglong Chen

With Thanks to Dr. Xin Feng and Dr. Haipeng Guo

With courtesy to Dr. Judy Feng of UIC

Outline

- Run arguments
- File reading
- File writing

Data Input

- Data input
  - scanf
  - gets
  - command line parameters
  - file

Command Line Parameters

the number of arguments passed

each element points to an argument

```
#include <stdio.h>

int main(int argc, char *argv[]) {

   int i;

   printf("There are %d command line parameters. They
   are:\n", argc);

   for (i = 0; i < argc; i++)

       printf("%s\n", argv[i]);

   return 0;

}
```

Limitation

- Use scanf or the command line input, we can only input a small amount of data, the data is from keyboard
- For large volume of data, we need to use File I/O. This is also another resource of data.

Data Input

- Resource of data
  - keyboard
    - a small amount of data
  - file
    - A large number of data

Basic File Operations

- Open a file
- Read data from a file
- Write data to a file
- Close a file

Declare a File Pointer

- Format
  - `FILE *fp;`
- declares a pointer variable that points to FILE type.
- reading or writing files is through the defined pointer variables.

Open a File: fopen

- Prototype: `FILE* fopen(char *fileName, char *mode);`
- Function call

File name

Open mode

```
fp = fopen(fileName, mode);
```

Declared file pointer variable

- If the file fails to open, `fopen` returns `NULL`
- If the file opens successfully, it will return a file pointer to the pointer variable.

Open a File - mode

```
fp = fopen(fileName, mode);
```

"r"	Open a file for reading. The file must exist.
"w"	Create an empty file for writing. If a file with the same name exists, its content is erased and the file is treated as a new
"a"	Append to a file. Append data at the end of the file. The file is created if it does not exist.
"r+"	Open a file for update both reading and writing. The file must
"w+"	Create an empty file for both reading and writing. If a file with same name already exists, its content is erased and the file is as a new empty file.
"a+"	Open a file for reading and appending.

An Example

Format: `fp = fopen(fileName, mode);`

```
FILE *myFile;
myFile = fopen("data.ini", "r");
```

Close a File: fclose

- Prototype: `int fclose(FILE* filePointer);`
- Function call

```
fclose(fp);
```

- close the file associated with pointer variable `fp`.
- e.g.,

```
FILE *myFile;
myFile = fopen("data.ini", "r");
...
fclose(myFile);
```

Read From/Write To A File

- Functions
  - `fgetc()`
  - `fputc()`
  - `fgets()`
  - `fputs()`
  - `fscanf()`
  - `fprintf()`

fgetc

- Prototype: `int fgetc(FILE* filePointer);`

char variable

- Function call: `c = fgetc(fp);`

file pointer variable

- reads a character from the file associated with `fp`
- if `fp` reaches the end of the file, the character is `EOF` (means the end of file)

fputc

- Prototype: `int fputc(char c, FILE* filePointer);`

char variable or constant

- Function call: `fputc(c, fp);`

file pointer variable

- write `c`'s value into the file associated with `fp`

An Example

```
FILE *fp;
char c;
fp = fopen("infile.txt", "r");
while((c = fgetc(fp)) != EOF) {
  printf("%c", c);
}
fclose(fp);
```

fgets

- Prototype: `char* fgets(char* str, int size, FILE* filePointer);`

String variable

- Function call: `fgets(str, size, fp);`

Number of characters

file pointer variable

- read a string to `str` with the length `size` or a line from the file associated with the `fp`
- when reach the end of the file, return `NULL`

fputs

- Prototype: `int fputs(char* str, FILE *filePointer);`

string
Variable or constant

- Function call: `fputs(str, fp);`

file pointer variable

- write the string into the file associated with `fp`

An Example

```
FILE *src, *dst;
  char str[256];
  src = fopen("infile.txt", "r");
  dst = fopen("outfile.txt", "w");
  while((fgets(str, 256, src)) != NULL){
     fputs(str, dst);
  }
  fclose( src );
  fclose( dst );
```

fscanf

- Prototype: `int fscanf(FILE *filePointer, const char *format, ...);`

- Function call: `fscanf(fp, format, ...);`

file pointer variable

Same as in scanf

- read data in the designated format from a file associated with `fp` like `scanf` from keyboard
- when reach the end of the file, return `EOF`

Class Exercise #1

1) Create a text file infile.txt

```
Jerry 20
Jim 10
Tony 12
```

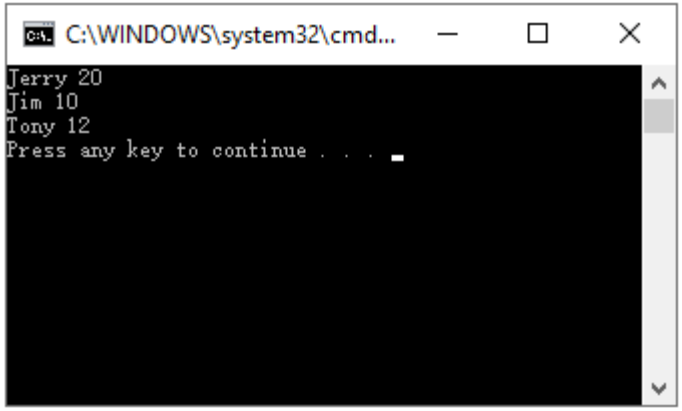
2) Compile and run the program

```c
#include <stdio.h>
int main()
{
   FILE *fp;

   char stuName[20];
   int stuID;
   fp = fopen("infile.txt", "r");
   while(fscanf(fp,"%s %d",stuName,&stuID)!= EOF)
   printf("%s %d\n", stuName, stuID);

   fclose(fp);
   return 0;
}
```

3) Output



fprintf

- Prototype: `int fprintf(FILE *filePointer, const char *format, ...);`

- Format: `fprintf(fp, format, ...);`

file pointer variable

Same as in printf

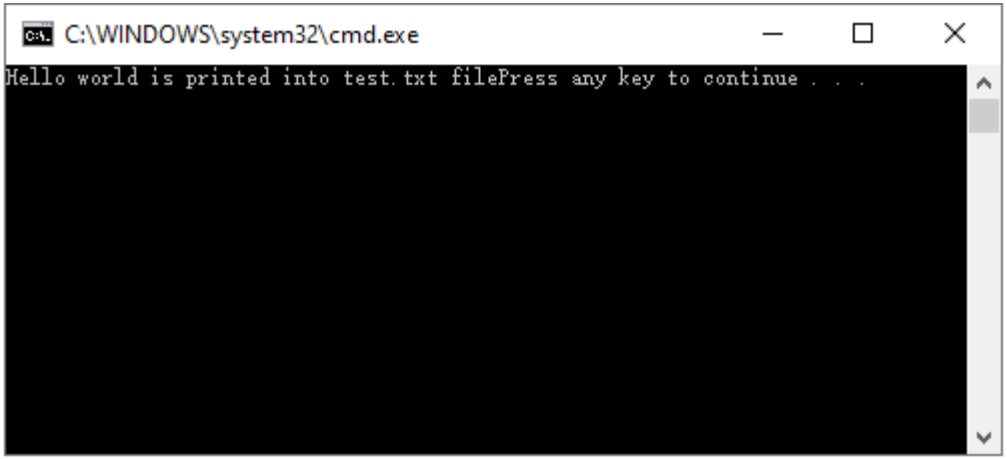
- Write data in the designated format from a file associated with `fp` like `printf` to monitor.

Class Exercise #2

1) Compile and run the program

```
#include <stdio.h>
int main()
{
  FILE *fp;
  fp = fopen("test.txt", "w");
  if (fp == NULL) {
  printf("Error: can't create file.\n");
  return 1;
  }else{
    fprintf(fp, "%s", "Hello world!\n");
    printf("Hello world is printed into test.txt file");
  }
  fclose(fp);
  return 0;
}
```

2) Output



Class Exercise #3

1) Compile and run the program

```
#include <stdio.h>
  int main() {
  int i;
  FILE *outfile;

  outfile = fopen("out.dat","w");
  for (i=0;i<10;i++) {
    fprintf(outfile,"%d\n",i+1);
  }
  fclose(outfile);
  return 0;
}
```

2) The out.dat should look like this:

out.dat × File#1.c Lec1.cpp* Solution Explorer

```
1
2
3
4
5
6
7
8
9
10
```

3) Modify the program to calculate and store Fibonacci sequence into out.dat Like this (first 10, start with 1):

out.dat × File#1.c Lec1.cpp Solution Explorer

```
1
2
3
5
8
13
21
34
55
89
```

After Class

- How to open a file in binary?
- More file functions
  - `fread()` // Read block of data from stream
  - `fwrite()` // Write block of data from stream
  - `ftell()` // returns the current file position of the given stream.
  - `rewind()`

  /* Sets the position indicator associated with *stream* to the beginning of the file. */

Summary

- Introduced how to read and write a file
- File handling is very important in information handling
- File stream pointer is used in reading and writing

Structured Programming

- Dynamic Memory Allocation

With courtesy to Dr. Judy Feng of UIC

Outline

- Static memory allocation and dynamic memeory allocation
- Memory allocation functions

Memory Allocation

- Two ways
  - static memory allocation
    - Memory is allocated invisibly when a variable is declared
      - e.g., `int i;` four bytes are allocated to store the value of i.
  - dynamic memory allocation
    - Memory is allocated visibly by calling some allocation functions.

Problems with Static Memory Allocation

- Memory allocation determined at compiling time:
  - `int x;` // 4 bytes
  - `char y;` // 1 byte
  - `float a;` // 4 bytes
  - `double b;` // 8 bytes
  - `int a[10];` // 10*4 bytes
- The type of the variable determines how much memory the compiler allocates

Static Memory Allocation

- If too much memory is allocated and then not used, there is a waste of memory.
- If not enough memory is allocated, the program is not able to handle the input data.

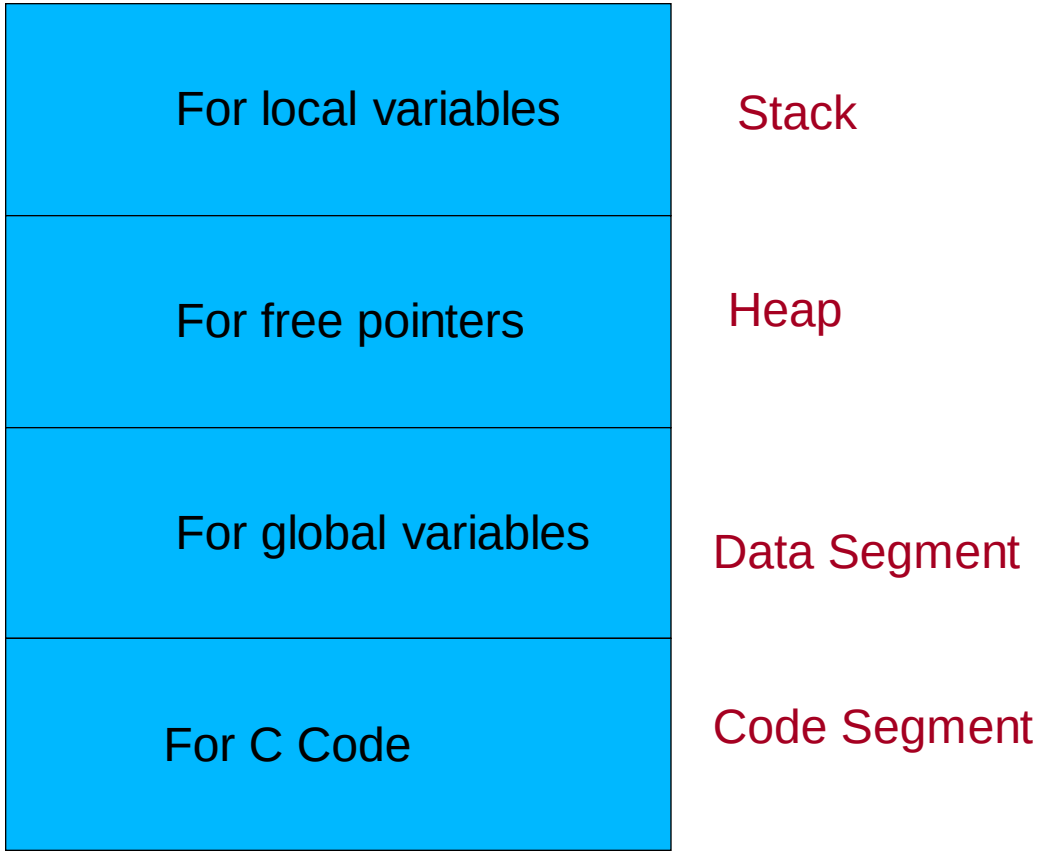
```
int grade[100];
int i = 0;
scanf("%d", &grade[i]);
while(grade[i] != -1)
{
  printf("%d\n", grade[i]);
  i++;
  scanf("%d", &grade[i])
}
```

What if there are only 5 students?
What if there are 1000 students?

Dynamic Memory Allocation

- Dynamic memory allocation allocates memory at execution time or runtime when needed, free the memory when the memory is not needed
- Dynamic memory is allocated in the heap (also called the free storage, a large pool of unused memory area) by the system.
- Static memory is allocated in the stack (last-in-first-out) the system.

Memory Allocation Process



Dynamic Memory Allocation Functions

- Two basic functions
  - `void *malloc(long size)`
    - Allocates a block of memory of specified size and returns a pointer of type void
    - We can assign it to a pointer of any type
    - A null pointer is returned if there is not enough space in the heap
  - `void free(void *ptr)`
    - releases the used memory when it is no longer needed
- To use these functions, the stdlib.h header file must be included.

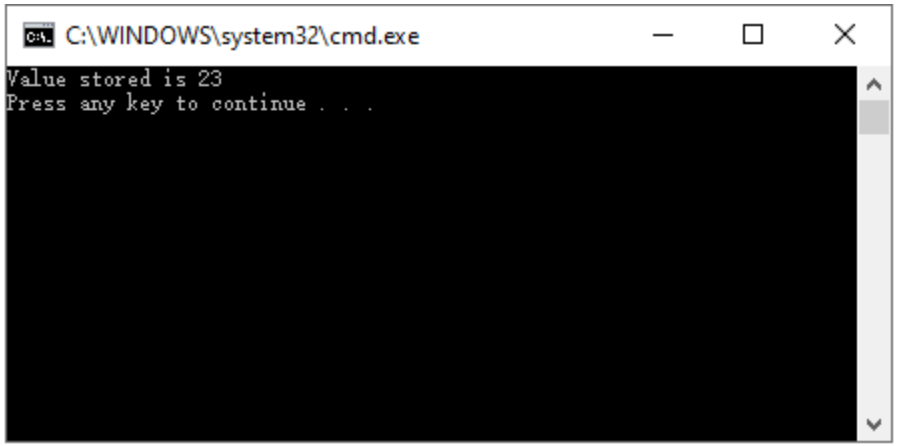
Class Exercise #4

1) Compile and run the program

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int *ptr;
ptr = (int*)malloc(sizeof(int));
if (ptr != NULL)
*ptr = 23;
printf("Value stored is %d\n", *ptr);
free(ptr);

return 0;
}
```

2) Output



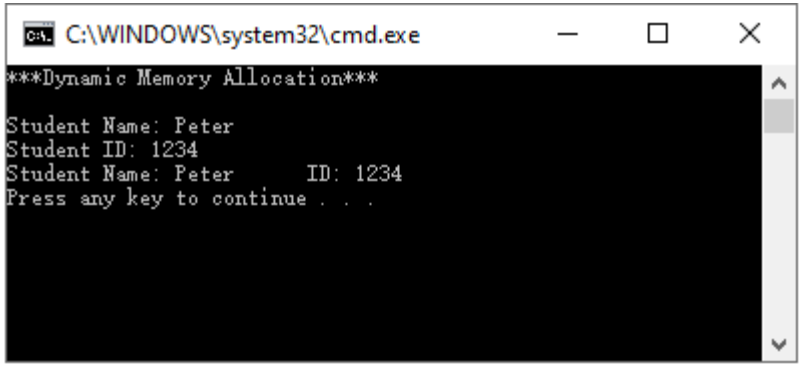
Class Exercise #5

1) Compile and run the program

```
#include <stdio.h>
#include <stdlib.h>
struct stdrec{
char name[20];
int id;
};

int main() {
  struct stdrec *p;
  printf("***Dynamic Memory Allocation***\n\n");
  p = (struct stdrec *)malloc((sizeof(struct stdrec)));
  if (p){
    printf("Student Name: ");
    gets(p->name);
    printf("Student ID: ");
    scanf("%d%*c", &p->id);
    printf("Student Name: %-10s ",p->name);
    printf("ID: %4d\n", p->id);
  }else{
    printf("Out of memory\n");
  }
return 0;
}
```

2) Output should like this



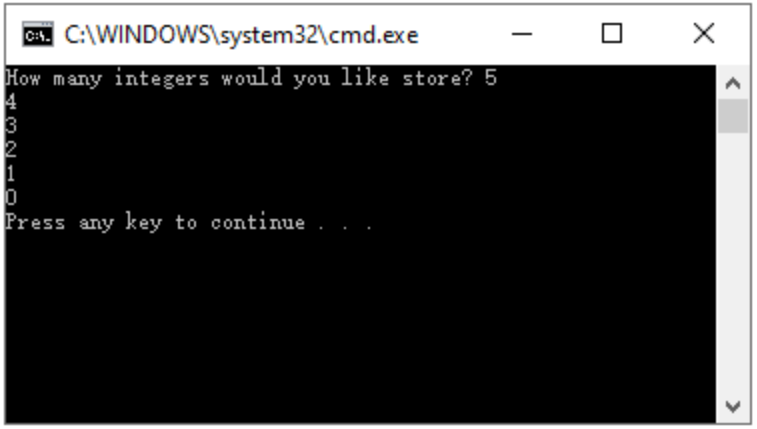
Class Exercise #6

1) Compile and run the program

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
  int number, i;
  int *ptr;

  printf("How many integers would you like store? ");
  scanf("%d", &number);
  ptr = (int*)malloc(number * sizeof(int));
  if(ptr != NULL){
    for(i = 0; i < number; i++){ *(ptr + i) = i; }
    for(i = number; i > 0; i--)
      printf("%d\n", *(ptr + (i - 1)));
    free(ptr);
   }
  return 0;
}
```

2) Output should like this



3) Using the student record struct in Ex 5 Design C program to enter and print (says) 10 student records using memory allocation technique.

More Functions

- void *calloc(int n, int elem-size)
  - allocates n blocks of storage, each of the same size (elem-size), and then sets all bytes to zero
  - A null pointer is returned if there is not enough space
- void *realloc(void *ptr, int newsize)
  - allocates a new memory space of size newsize to the pointer variable ptr
  - A null pointer is returned if there is not enough spac
  - The newsize maybe larger or smaller than the old size

https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/

Memory Leak

- Memory leak happens when memory is allocated but not released
- It will cause an application to gradually consume memory then the available memory for other applications is reduced.

The allocated memory must be freed!!!

Summary

- Dynamic memory allocation can save memory when data size is un-determined.
- Functions can be used to allocate and free memory