GNG1106
Fundamentals of Engineering Computation

Instructor: **Hitham Jleed**

University of Ottawa

Fall 2023 ~

In-Class Exercise:

Outline

Binary File

- A binary file is simply a sequence of bytes that have been copied directly from memory; a binary file is in effect a "memory dump".
- Advantages of a binary file
  - smaller than an ASCII file
    - e.g. 121232444 stored in an ASCII file takes 9 bytes but stored in a binary file (as an int-typed value) takes 4 bytes
  - faster read/write than an ASCII file
    - e.g., a real number, read in as a sequence of bytes in an ASCII file (where each byte represents a digit or the decimal point), must be converted using %f to a float before it is stored in a variable of type float. Such a conversion is not necessary when reading from a binary file, since the binary representation of the real number in the file can be directly loaded into a float-type variable.
- Disadvantages of a binary file
  - It cannot be read/edited using a text editor
  - Limited portability across machines/OS's

Write to Binary File: fwrite

```
FILE *fp;
int integer = 100;
float real = 7.5;
double dbl[3]={1.0, 1.1, 1.2};
fp = fopen("test.bin", "wb");
fwrite(&integer, sizeof(int), 1, fp);
fwrite(&real, sizeof(float), 1, fp);
fwrite(dbl, sizeof(double), 3, fp);
fclose(fp);
```

- 1st parameter: the address of the memory storing the data
- 2nd parameter: the size (in bytes) of each data element
- 3rd parameter: the number of data elements to write
- 4th parameter: the FILE pointer.

Read from Binary File: fread

```
FILE *fp;
int integer;
float real;
double dbl[3];
fp = fopen("test.bin", "rb");
fread(&integer, sizeof(int), 1, fp);
fread(&real, sizeof(float), 1, fp);
fread(dbl, sizeof(double), 3, fp);
fclose(fp);
```

- 1st parameter: the address of the memory to store the read data
- 2nd parameter: the size (in bytes) of each data element
- 3rd parameter: the number of data elements to read
- 4th parameter: the FILE pointer.

Return of functions `fwrite` and fread

- `fwrite` returns the number of elements successfully written to the file.
  - If a writing error occurred, the number returned by `fwrite` will be smaller than the value of the third argument in its argument list.
- `fread` returns the number of elements successfully read from the file.
  - If a reading error occurred or if EOF was detected, the returned value will be smaller than the value of the third argument in its argument list.

Outline

Function Returning a Pointer

```
someType *myFunction(...)
{
  someType *ptr;
   ...
  ptr=(someType*)malloc(K*sizeof(someType));
  // K is an integer whose value has been resolved
   ...
  // assign value to ptr[0], ptr[1], ... ptr[K-1]
   ...
  return ptr;
}
```

Highlight

The memory allocated by `malloc` and pointed to by `ptr` lives in the heap. It stays allocated after the function exists.

- When a function returns a pointer, the pointer must not point to its local variables.
- Such an exercise is actually calling for illegal memory access! The following two functions will mostly likely result in run-time error (but they will pass compiler!), since the memory pointed to by `p` (which is in the stack) will be released after the function returns. If the returned address `p` from the function is used for accessing that block of memory, such an access would be illegal.

```
int *WrongFunction()
{
  int a;
  int *p;
  ...
  p=&a;
  return p;
}
```

```
int *WrongFunction()
{
  int a[2]={2, 3};
  int *p;
  ...
  p=a;
  return p;
}
```

Pointer to Pointer

Declaring a pointer to pointer:

```
int **x;
```

- Recall that "`int *`" can be regarded as a type.
- Then "`int **`" can be regarded as another type.
- This declaration statement indicates the following.
  - The memory block, say A, allocated for variable `x`, will be used to store an address. That is, variable `x` is address-typed.
  - The memory block, say B, pointed to by the value of `x` will be used to store one or several addresses.
  - The memory block pointed to by each address stored in B will be used to store an `int`-typed value.
- This declaration statement allocates memory A only.

- More generally, a pointer to to pointer is declared by

```
someType **x;
```

- A popular use of pointer to pointer is to implement 2D arrays of arbitrary sizes, which offers great flexibility comparing with using 2D arrays.
- It can also be used more generally to implement a list of arrays of varying lengths.
- When a pointer, say `int *x`, needs to be passed into a function and the function must change the value of `x` after it returns, we will have to pass the address of `x` into the function. Then the function must take an input variable that has type "`int **`".
- Example: implement a matrix as a pointer to pointer.

Coding Demonstration

Outline

Solving problems with computing requires

- Understanding the problem
- Understanding the problem domain
- Understanding computing

Developing a large software often involves (iterations over) the following steps

1. Gather requirements (the customer's view)
2. Requirement specification (the developer's view)
3. Analysis and design
   - Is the objective feasible?
   - What design/core algorithm/technology will makes it feasible?
   - How to organize the software into modules, submodules, libraries etc? What kinds of data structure are needed?
4. Coding
5. Testing and integration
6. Deployment
7. Operation and Maintenance

The deliverables of a software usually includes

- the program
- internal documents
  - specifications of design and implementation
  - allowing better maintenance of the software
- external documents
  - report/white paper of the software
    - "look, this software is what you want me to build!"
  - installation/deployment instruction
  - user manual, etc.