CH-230-A

Programming in C and C++

C/C++

Tutorial 8

Dr. Kinga Lipskoch

Fall 2020

Reading/Writing

Prototype	**Use**
`int getc(FILE *fp)`	Returns next `char` from `fp`
`int putc(int c, FILE *fp)`	Writes a `char` to `fp`
`int fscanf(FILE* fp, char * format, ...)`	Gets data from `fp` according to the format string
`int fprintf(FILE* fp, char * format, ...)`	Outputs data to `fp` according to the format string

Line Input and Line Output

```
char *fgets(char *line, int max, FILE *fp);
```

- Already seen with `stdin`
- Used for files as well

```
int fputs(char *line, FILE *fp);
```

- Outputs/writes a string to a file

Files: Example 1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main() {
4    char ch;
5    FILE *fp;
6    fp = fopen("file.txt", "r");
7    if (fp == NULL) {
8      printf("Cannot open file!\n");
9      exit(1);
10   }
11   ch = getc(fp);
12   while (ch != EOF) {
13     putchar(ch);
14     ch = getc(fp);
15   }
16   fclose(fp);
17   return 0;
18 }
```

Files: Example 2

```
1  # include <stdio.h>
2  # include <stdlib.h>
3  int main () {
4    char ch;
5    FILE * fp;
6    fp = fopen("file.txt", "r") ;
7    if (fp == NULL) {
8      printf("Cannot open file!\n");
9      exit(1);
10   }
11   while((ch=getc(fp))!=EOF) {
12     putchar(ch);
13   }
14   fclose(fp);
15   return 0;
16 }
```

Files: Example 3

```
1  # include <stdio.h>
2  # include <stdlib.h>
3  int main () {
4    char ch;
5    FILE * fp;
6    fp = fopen("file.txt", "r") ;
7    if (fp == NULL) {
8      printf("Cannot open file!\n");
9      exit(1);
10   }
11   while(!feof(fp)) {
12     ch=getc(fp);
13     if (ch!=EOF)
14       putchar(ch);
15   }
16   fclose(fp);
17   return 0;
18 }
```

fflush(), feof(), ferror()

- `int fflush(FILE *stream)` flushes the output buffer of a stream
  - fflush_ex.c
- `int feof(FILE *stream)` tests the end-of-file indicator for the given stream
  - feof_ex.c
  - myfile.txt
- `int ferror(FILE *stream)` tests the error indicator for the given stream
  - ferror_ex.c

`fseek()` and `ftell()`

- Enables to use a file just like an array and move directly to a specific byte in a file that has been opened via `fopen()`
- `ftell()` returns current position of file pointer as a `long` value

`fseek(fp, offset, mode)`

- `fp` is a file pointer, points to file via `fopen()`
- `offset` is how far to move (in bytes) from the reference point
- `mode` specifies the reference point

Mode	measure offset from
`SEEK_SET`	beginning of file
`SEEK_CUR`	current position
`SEEK_END`	end of file

Examples

- `fseek(fp, 0L, SEEK_END);`
  - set position to offset of `0` bytes from file end therefore set position to end of file
- `long last = ftell(fp);`
  - assigns to `last` the number of bytes from the beginning to end of file

Binary I/O

- `fread()` and `fwrite()`
- Standard I/O is text-oriented
  - Characters and strings
- How to save a double
  - Possible as string but also other

```
double num = 1/3.0;
fprintf(fp, "%lf", num);
```

- Most accurate way would be to store the bit pattern that program internally uses
- Called binary when data is stored in representation the program uses

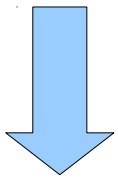
I/O as Text

- All data is stored in binary form
- But for text, data is interpreted as characters

```
short int num = 12345      // a 16-bit number
```

stores 12345 as binary number in num

00110000	00111001



fprintf(fp, "%d", num);

writes binary code for
characters '1', '2', '3', '4', '5' to file

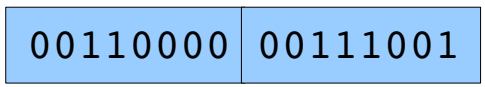
00110001	00110010	00110011	00110100	00110101

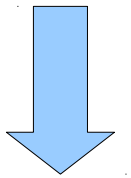
I/O as Binary

If data is interpreted as numeric data in binary form, data is stored as binary

```
short int num = 12345     // a 16-bit number
```

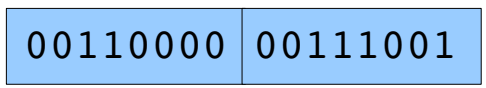
stores 12345 as binary number in num





fwrite(&num, sizeof(short int), 1, fp);

writes binary code the value 12345 to file



fwrite() (1)

- `size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *fp)`
- Writes binary data to a file
- `size_t` - type is type that `sizeof()` returns, typically `unsigned int`
- `ptr` - address of chunk of data to be written
- `size` - size in bytes of one chunk
- `nmemb` - number of chunks to be written
- `fp` - file pointer to write to

fwrite() (2)

```
1 char buffer[256];
2 fwrite(buffer, 256, 1, fp);
```

- Writes 256 of bytes to the file

```
1 double price[10];
2 fwrite(price, sizeof(double), 10, fp);
```

- Writes data from the price array to the file in 10 chunks each of size double
- Return number of items successfully written, may be less if write error

fread()

- `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *fp)`
- Takes same set of arguments that `fwrite()` does
- `ptr` pointer to which data is read to

```
1 double price[10];
2 fread(price, sizeof(double), 10, fp);
```

- Reads `10` size double values into the `price` array
- Returns number of items read, maybe less if read error or end of file reached

Programming Environment (1)

- C++ is available on practically every operating system
- Commercial
  - Microsoft C++, Intel C++, Portland
- As well as free compilers available
  - g++
- g++ available on many platforms
- g++ `8.3.0` is used on Grader

Programming Environment (2)

- We will refer to the Unix operating system and related GNU tools (`g++`, `gdb`, some editor, IDE, etc.)
- Install a C++ compiler on your notebook
- IDE (Integrated Development Environment) may be helpful
- Code::Blocks
- Visual Studio Code
- Visual Studio Community 2019
- Make sure that you have `g++` installed

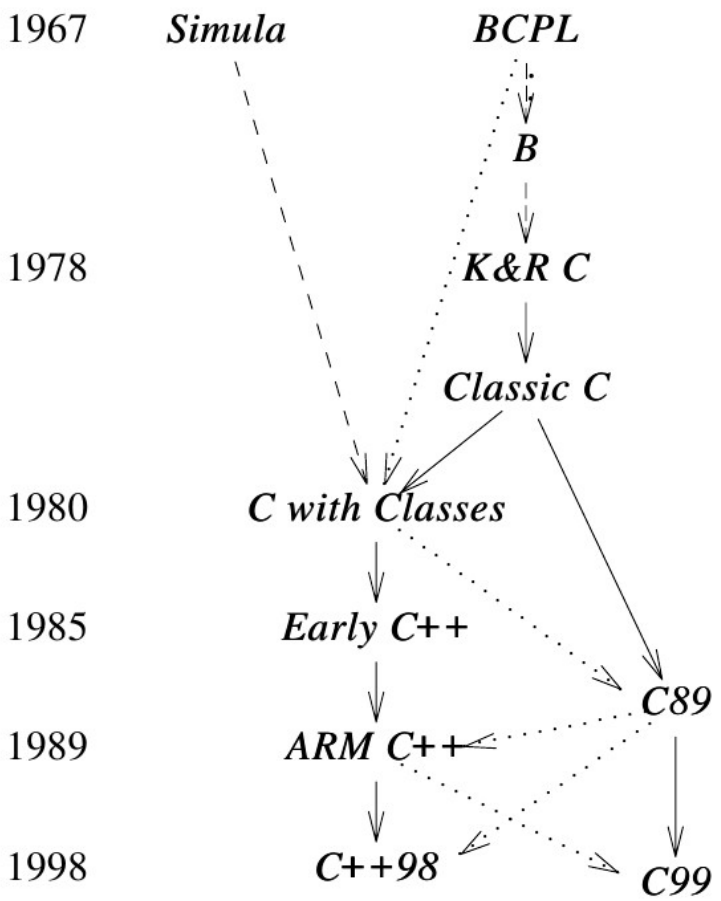
Brief History of C++

C++ is an object-oriented Extension to C

- 1970's B. Kernighan and D. Ritchie create C language at Bell Labs
- 1980's Bjarne Stroustrup creates C successor
  - C with classes
  - Using some pre-compiler C++ code was translated into C and compiled then
- 1998 C++ finally becomes ISO standard (ISO/IEC 14882:1999)
- 2003 C++03 (ISO/IEC 14882:2003)
- 2005 Technical Report (TR1) (many extensions)
- 2011 C++11 (ISO/IEC 14882:2011)
- 2014 C++14 (ISO/IEC 14882:2014(E))
- 2017 C++17 (final standard published in December)

The Way to C++

- Simula has been developed in the 1960s at the Norwegian Computing Center in Oslo, by Ole-Johan Dahl and Kristen Nygaard
- Based on ALGOL 60 it is considered to be the first object-oriented programming language
- Here ARM is not the now more and more famous processor, but Bjarne Stroustrup's ARM (Annotated Reference Manual) which was the de-facto standard, since no official standard existed



C++

C is almost a complete subset of C++

- possibility to clean up some things, but
- compatibility to millions lines of code considered more important
- some differences remain
- char constants:
  - 1 byte in C++ (actually `char`)
  - 4 bytes in C (actually an `int`)
- older C++ standards required prototypes (from standard C11 no implicit need for prototypes)

C++ and OOP

- C++ is an improved version of C, which includes constructs supporting Object-Oriented Programming (OOP)
  - Improved version of C means that
    - Almost all the code written in C will be compiled by a C++ compiler
    - C++ fixes some C "holes"
    - This may need minor code adjustments
    - Libraries written for C can be used for C++
    - The meaning of C constructs is not altered
- C++ is a multi-paradigm language
- It does not force you to exclusively use OOP, but it lets you mix OOP and classic imperative programming (differently from other pure OO languages like Java, Smalltalk, etc.)

What is OOP?

- OOP is a programming paradigm, i.e., a way to organize the solution of your computational needs
  - But definitely not the only way
- Pros and cons
  - Increases productivity at very different levels (code reuse, generic programming, hopefully simpler design process)
  - OOP is not the solution for every need
  - OOP could be not easy to learn and takes a long time before one can master it properly

Imperative Programming vs. OOP (1)

Imperative programming relies on a top-down approach

- Iteratively divide the given problems into simpler sub-problems (simpler from a logic point of view)
- When a sub-problem is simple enough, code it
  - Again, simple means simple from a logic point of view, and not from the point of view of the number of lines of code
- Interactions between sub-problems happen by mean of function calls
  - You have seen this while programming in C

Cornerstones of OOP

- Data abstraction (hiding of information)
- Encapsulation (hiding of internal workings)
- Inheritance (relation between class and subclass)
- Polymorphism (ability to use the same syntax for objects of different types)

Imperative Programming vs. OOP (2)

- OOP follows a bottom-up process
- Given a problem you should first ask yourself
  - Which are the entities which characterize this problem?
  - What are their characteristics? (member data)
  - How do they interact? (methods/functions)
  - Entities (objects) interact by mean of messages exchanged between them
    - A message is a request to execute a method

OOP Characteristics (by Alan Kay, inventor of Smalltalk)

1. Everything is an object
2. A program is a collection of objects exchanging messages
3. Each object has a memory made by other objects
4. Every object has a type
5. Objects of the same type can receive the same messages

The Early History of Smalltalk, Alan C. Kay ACM SIGPLAN Notices Volume 28, No. 3, March 1993 Pages 69–95

http://stephane.ducasse.free.fr/FreeBooks/SmalltalkHistoryHOPL.pdf