BASIC PROGRAMMING IN C

FILE PROCESSING

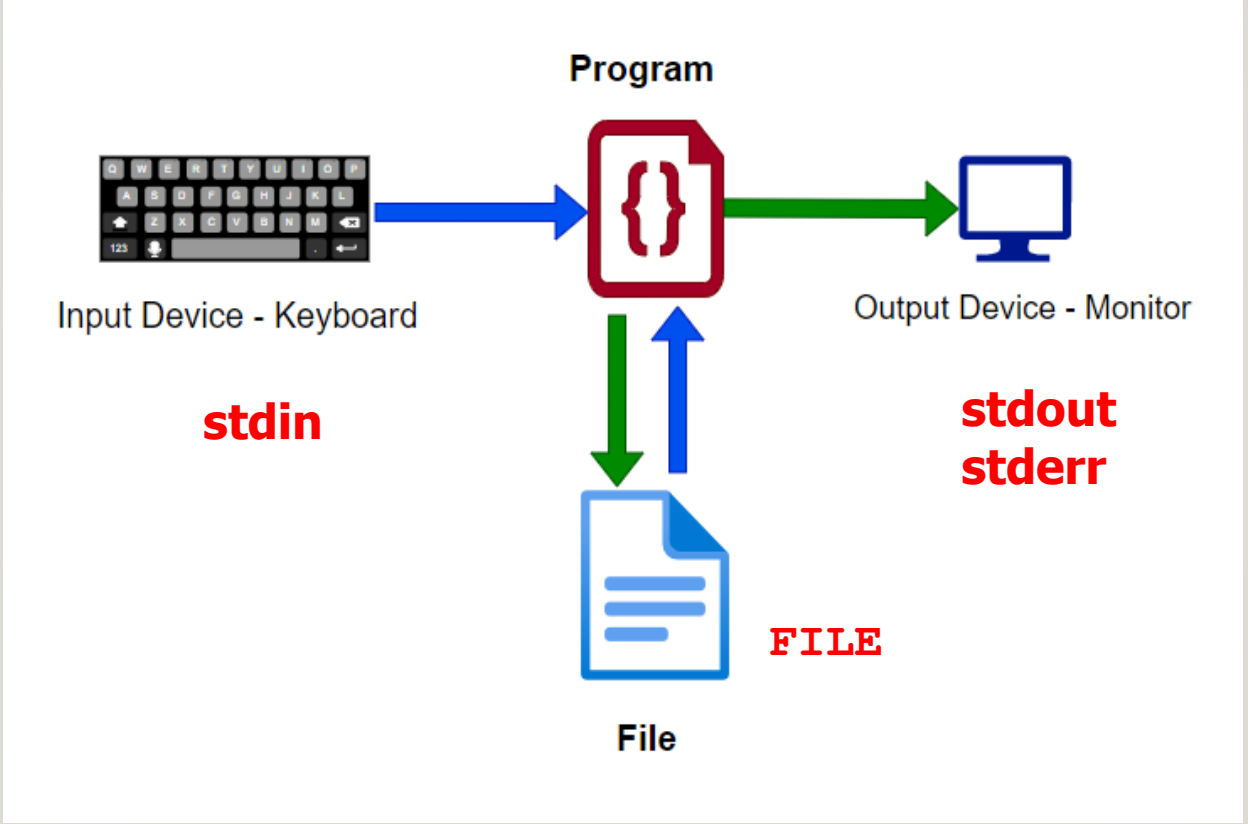
INDRA DWI RIANTO, S.KOM, S.SI, M.T.I. (INDRA.RIANTO@BINUS.EDU)

OUTLINE

- Files and Streams
- Buffer Area
- Open File
- Close File
- Input & Output File
- Program Examples
- Exercise

Materi Basic Programming in C - File Processing
http://bit.ly/3Guoopf

FILES AND STREAMS



Sumber Gambar

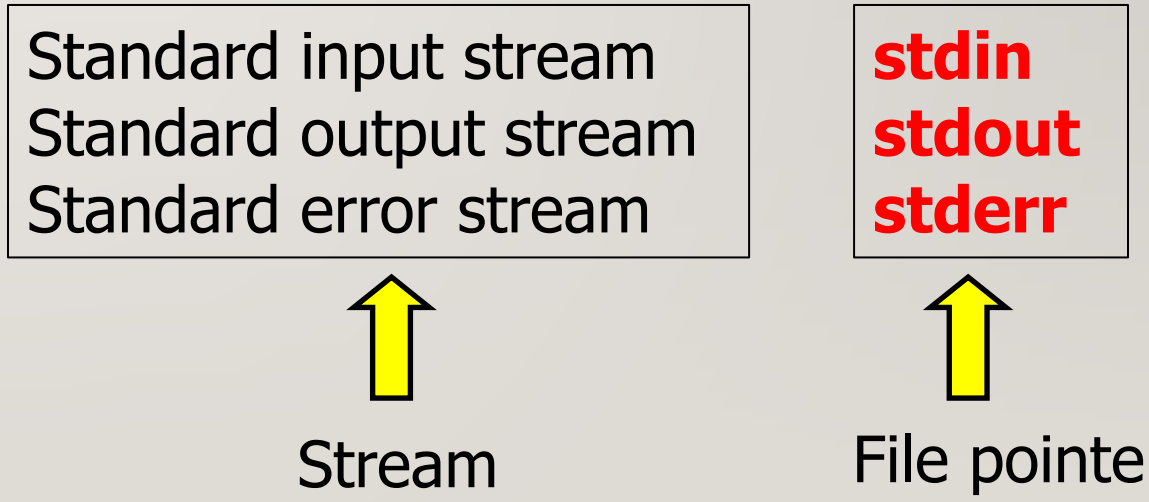
FILES AND STREAMS

When a C program run, there are three (3) standard streams activated:

1. **Standard Input Stream**
   Controlling input stream from keyboard
2. **Standard Output Stream**
   Controlling output stream to the monitor
3. **Standard Error Stream**
   Controlling the error messaging

Each stream associated with a file.

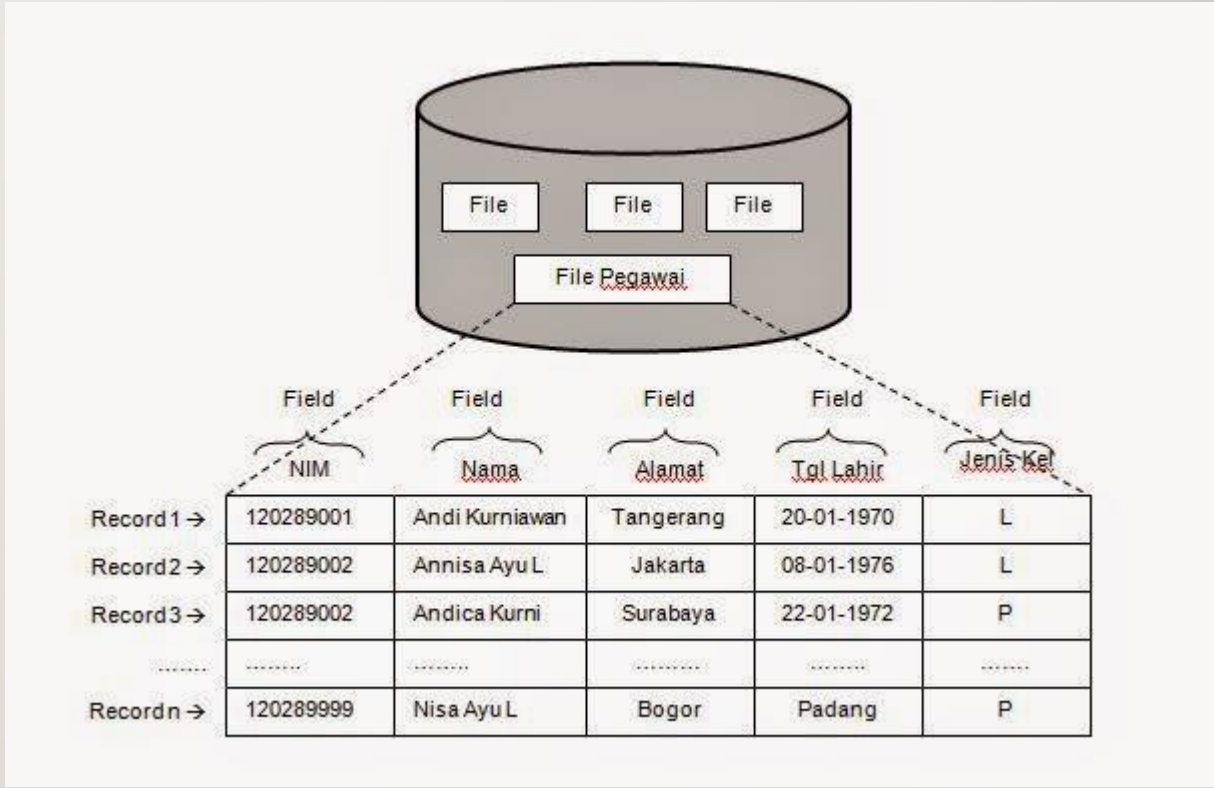
Opening a file ordering a pointer returned to the initiator. The Pointer is pointing to a data structure with **FILE** type defined in `stdio.h`



FILES AND STREAMS

File Definition

- File is a collection of record
- Record is a collection of field
- Field is a block of byte
- Byte is collection of bit



Sumber Gambar

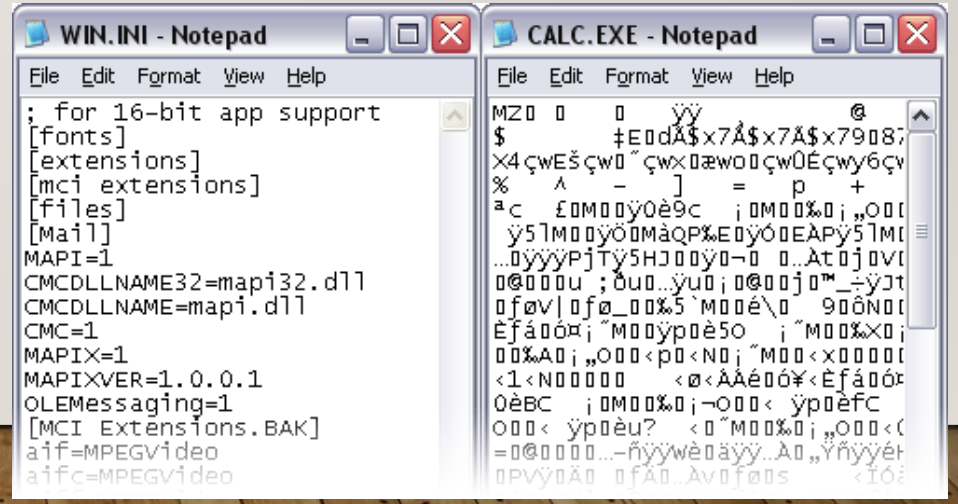
FILES AND STREAMS

```
typedef struct  {
   int level;                 // fill/empty level of buffer
   unsigned flags;            // File status flags
   char fd;                   // File descriptor
   unsigned char hold;        // Unget char if no buffer
   int bsize;                 // Buffer size
   unsigned char *buffer;     // Data transfer buffer
   unsigned char *curp;       // Current active pointer
   unsigned istemp;           // Temporary file indicator
   short token;               //Used for validity checking
}  FILE;
```

FILES AND STREAMS

TEXT FILE saved in a text format or **ASCII File**
–Storage size depends on its data: 10000 needs 5 byte
–Can be open using standard text editor application
–or c:>TYPE file_name

BINARY FILE storing numerical data in affixed format in line with micro-processor format definition (example: format sign-magnitude 2's complement).



Sumber Gambar

BUFFER AREA

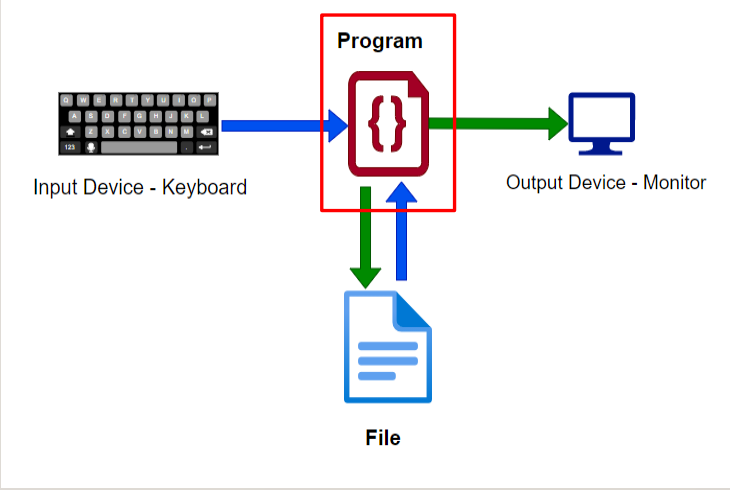
Buffer area is part of the memory used as a temporary space before data moved to a file.

Syntax:

FILE *fp;

Where fp is a file pointer pointing to the start of the buffer area.

Also known as stream pointer.



OPEN FILE

Opening a File using fopen():

```
FILE *fopen (const char *filename, const char *mode );
```

`fopen()` defined at **`<stdio.h>`**

`fopen()` return a pointer to the start of a buffer area. Null will be returned if file unable to open.

Possible mode value :

Mode	Description
"**r**"	opening a file to be read.
"**w**"	creating a file to be written.
"**a**"	opening a File for data append.
"**r+**"	opening a File for read/write.
"**w+**"	creating file for read/write.
"**a+**"	opening a File for read/append
"rb"	opening a File (binary) to be read.
"wb"	creating a file (binary) for write operation.

CLOSE FILE

Closing a File using fclose():

```
int fclose (FILE *stream);
```

`fclose()` defined at **`<stdio.h>`**

`fclose()` will return 0 if successful, and EOF if error

EOF (End Of File) equals to -1

`fclose()` will release the buffer area and immediately send the remaining data to file.

INPUT & OUTPUT FILE

fscanf **(INPUT)**

Syntax:

int fscanf(FILE *_stream_, const char *_format_ [, _argument_]...);

Read data from file inline with the scanf formatting.

Return the number of field read while successful, and EOF if error

fprintf **(OUTPUT)**

Syntax:

int fprintf(FILE *_stream_, const char *_format_ [, _argument_]...);

Writing data to a file using the printf format.

Return number of byte written if successful and negative value if error.

INPUT & OUTPUT FILE

fwrite

syntax: **size_t fwrite(const void *_buffer_, size_t _size_, size_t _count_, FILE *_stream_);**

Writing a block of data in the buffer area to the file

Return number of byte data written, and error otherwise.

fread

Syntax: **size_t fread(void *_buffer_, size_t _size_, size_t _count_, FILE *_stream_);**

Read a block size of data from a file

feof

Syntax : **int feof(FILE *_stream_);**

Finding out if the pointer has reached end-of-file

Return 0 if not end-of-file

PROGRAM EXAMPLES

- Example reading data from file test.txt using **fscanf**

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
      FILE *fp;  int no; char name[20]; float gpa;
      fp=fopen("test.txt","r");
      if(fp==NULL){
            printf("File test.txt can't be opened\n");
            exit(1);
      }
      fscanf(fp,"%d %s %f",&no,name, &gpa);
      printf("%d %s %f\n",no,name,gpa);
      fscanf(fp,"%d %s %f",&no,name, &gpa);
      printf("%d %s %f\n",no,name,gpa);
      fclose(fp);     return 0;
}
```

PROGRAM EXAMPLES

- Example writing data to file test.txt using **fprintf**

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *fp;
    fp=fopen("test.txt","w");
    if(fp==NULL){
        printf("File test.txt can’t be created\n");
        exit(1);
    }
    fprintf(fp,"%d %s %f\n",1,“Amir", 3.95);
    fprintf(fp,"%d %s %f\n",2,“Tono", 3.15);
    fclose(fp);
    return 0;
}
```

PROGRAM EXAMPLES

- Example reading data from binary file test.dat using **fread**

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *fp; int i;
    int Arr[5];
    fp=fopen("test.dat","r");
    if(fp==NULL){
        printf("File test.dat can't be opened\n");
        exit(1);
    }
    fread(Arr,sizeof(Arr),1,fp);
    for(i=0; i<5; i++) printf("%d ",Arr[i]);
    fclose(fp);
    return 0;
}
```

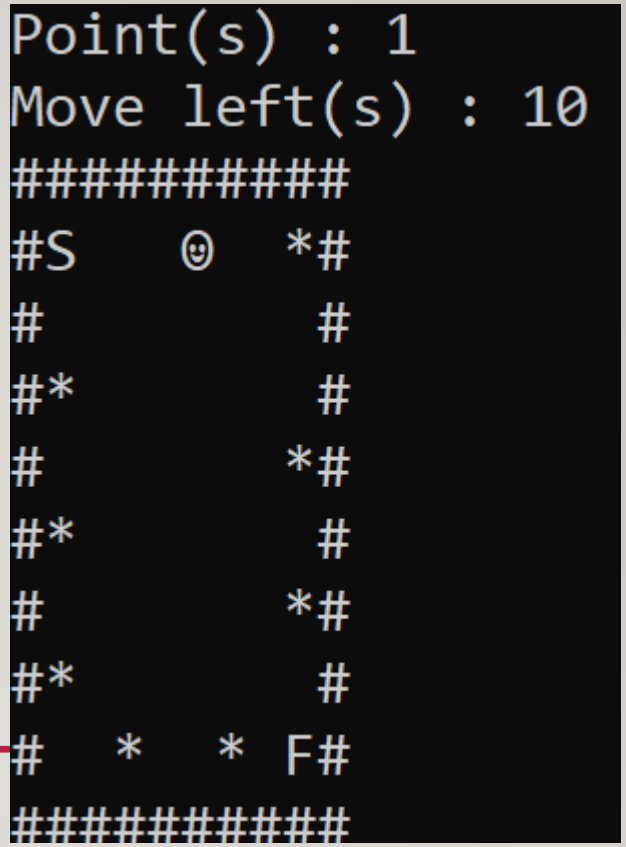
PROGRAM EXAMPLES

- Example writing data to binary file test.dat using **fwrite**

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *fp;
    int Arr[]={1,2,3,4,5};
    fp=fopen("test.dat","w");
    if(fp==NULL){
        printf("File test.dat can’t be created\n");
        exit(1);
    }
    fwrite(Arr,sizeof(Arr),1,fp);
    fclose(fp);
    return 0;
}
```

EXERCISE



Q N A