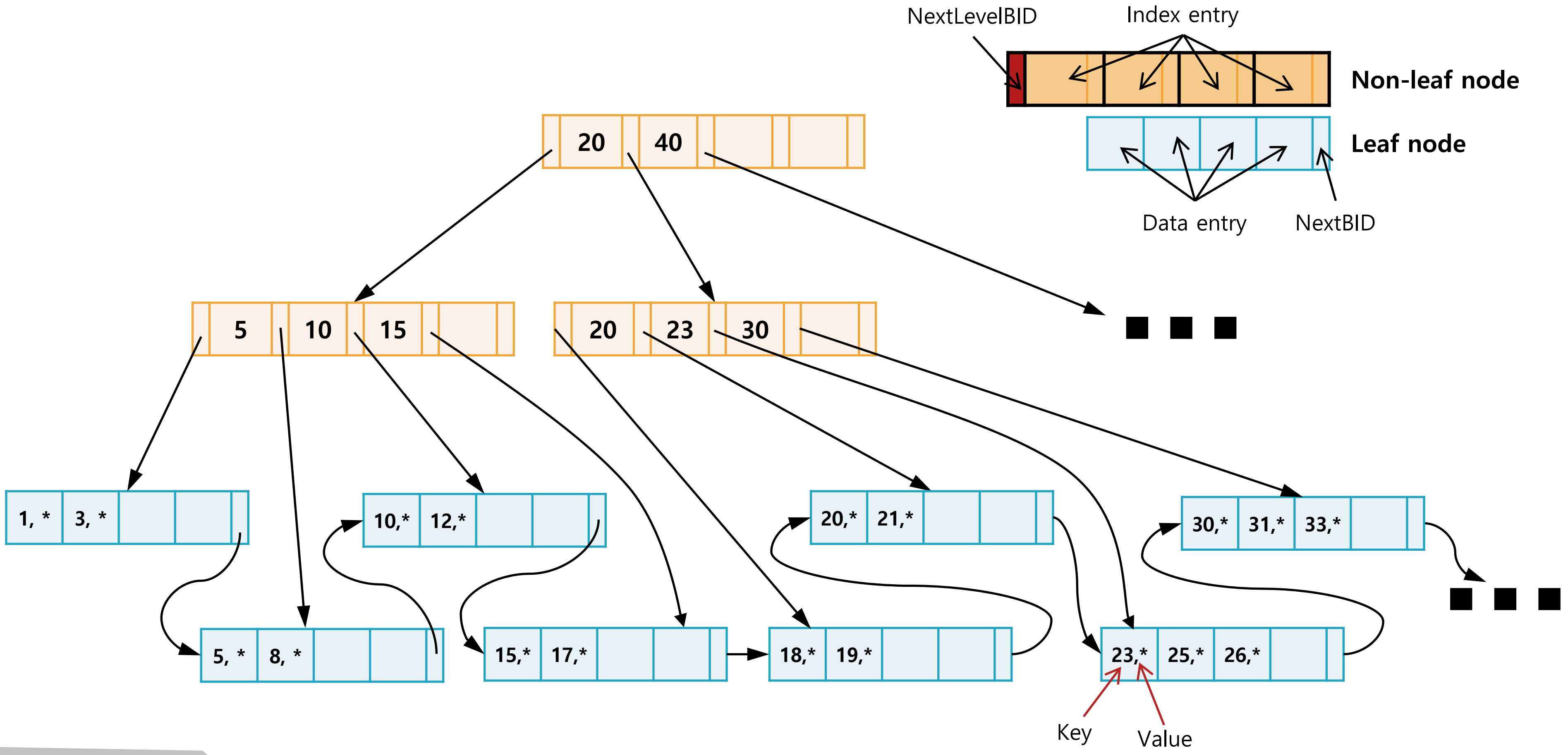
Implementation of a Disk Based B^+-Tree

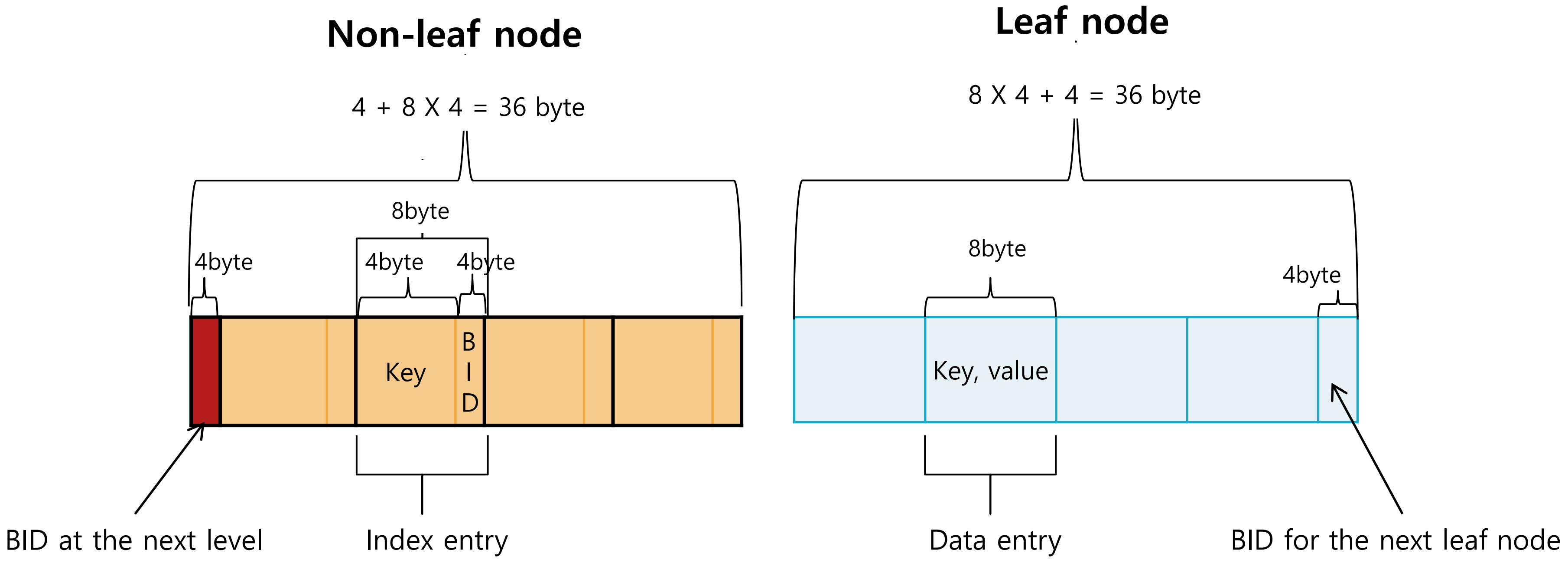
CSE3207 Database Project #2

Assignment Date : May 24th, 2021
Due Date : June 16th, 2021

B⁺-Tree Structure

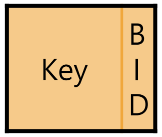


Details of Nodes and Entries

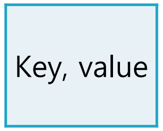


* block size = node size = 36 byte

Index entry
Key
NextLevelBID



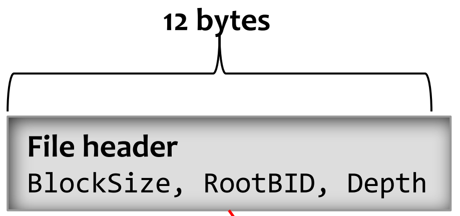
Data entry
Key
Value

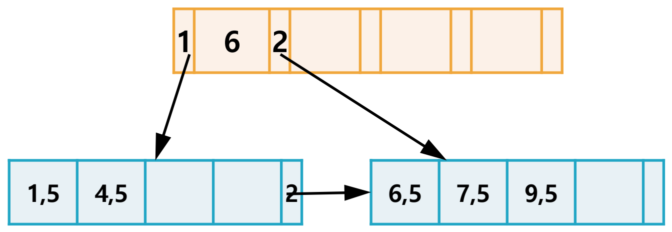


B⁺-Tree Data File Structure

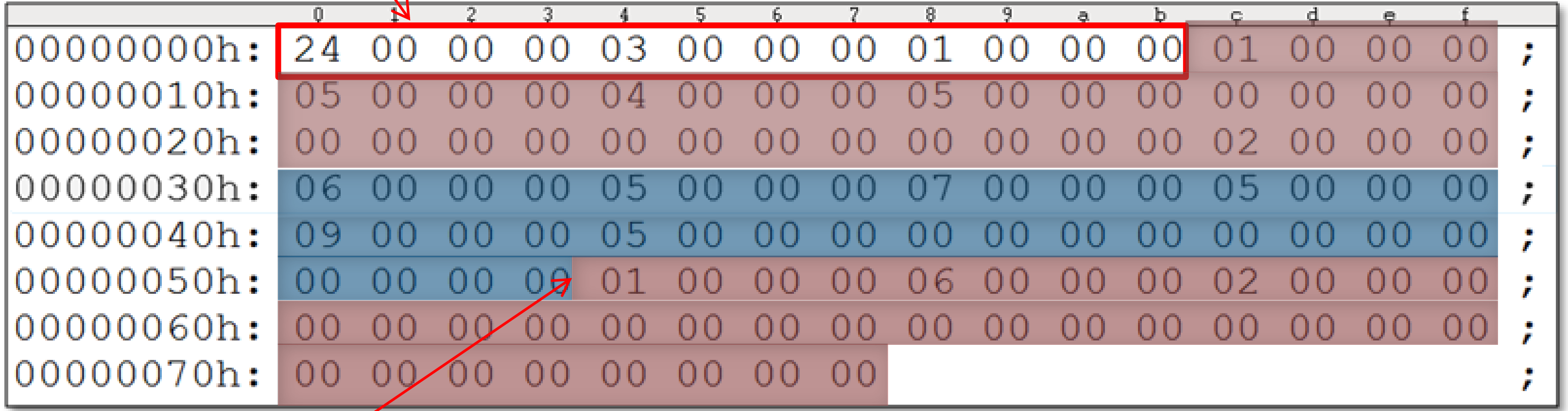
Insert data (key, ID)

1	5
6	5
4	5
7	5
9	5





***36 bytes for each node**



```
             0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 24 00 00 00 03 00 00 00 01 00 00 00 01 00 00 00 ;
00000010h: 05 00 00 00 04 00 00 00 05 00 00 00 00 00 00 00 ;
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00 ;
00000030h: 06 00 00 00 05 00 00 00 07 00 00 00 05 00 00 00 ;
00000040h: 09 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 ;
00000050h: 00 00 00 00 01 00 00 00 06 00 00 00 02 00 00 00 ;
00000060h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ;
00000070h: 00 00 00 00 00 00 00 00                         ;
```

84 bytes from the file starting position

*Physical offset of a Block ID = 12 + ((BID-1) * BlockSize)*

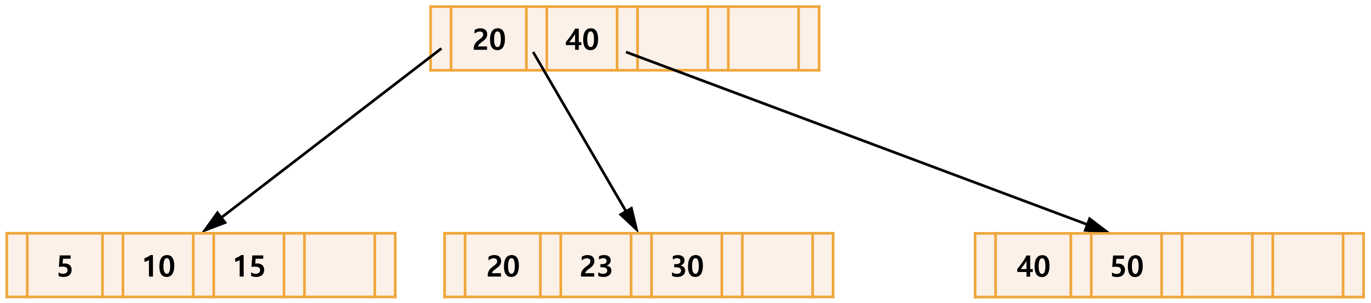
Test & UI

- Index creation
  - *btree.exe c [btree binary file] [block_size], e.g., btree.exe c btree.bin 36*
    - Generates [btree binary file] with only header
- Insertion
  - *btree.exe i [btree binary file] [records text file], e.g., btree.exe i btree.bin insert.txt*
    - Inserts nodes(entries) to [btree binary file] using [records text file]
- Point(exact) search
  - *btree.exe s [btree binary file] [input text file] [output text file],*
    e.g., btree.exe i btree.bin search.txt output.txt
    - Output searched keys and IDs to [output text file] using [btree binary file]
- Range search
  - *btree.exe r [btree binary file] [input text file] [output text file],*
    e.g., btree.exe r btree.bin rangesearch.txt output.txt
    - Output searched keys and IDs to [output text file] using [btree binary file]

- MUST follow input and output file formats in the document

Test & UI

- Print B⁺-Tree structure
  - *btree.exe p [btree binary file] [output text file]*
    - Output node structure of [btree binary file] to [output text file]
    - Output only root node<level 0> and next level <level1>
    - Example



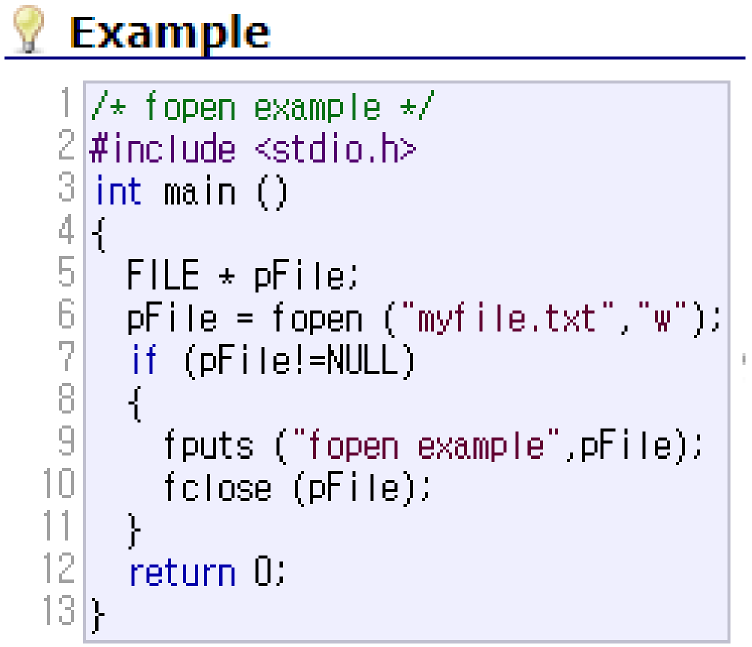
```
<0>
20, 40
<1>
5, 10, 15, 20, 23, 30, 40, 50
```

Submission

- To the I-Class website
- Upload a zip file containing the followings:
  - A single source file, named as ***“btree.cpp or btree.c”***
  - README.doc explaining:
    - What you’ve implemented and what you’ve NOT
    - Brief explanation of your implementation (Do not make it look fancy, less than 0.5 page)
    - How to compile and run
    - Talk about your experience of doing this project
    - Contact information (just in case)

C++ I/O library

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

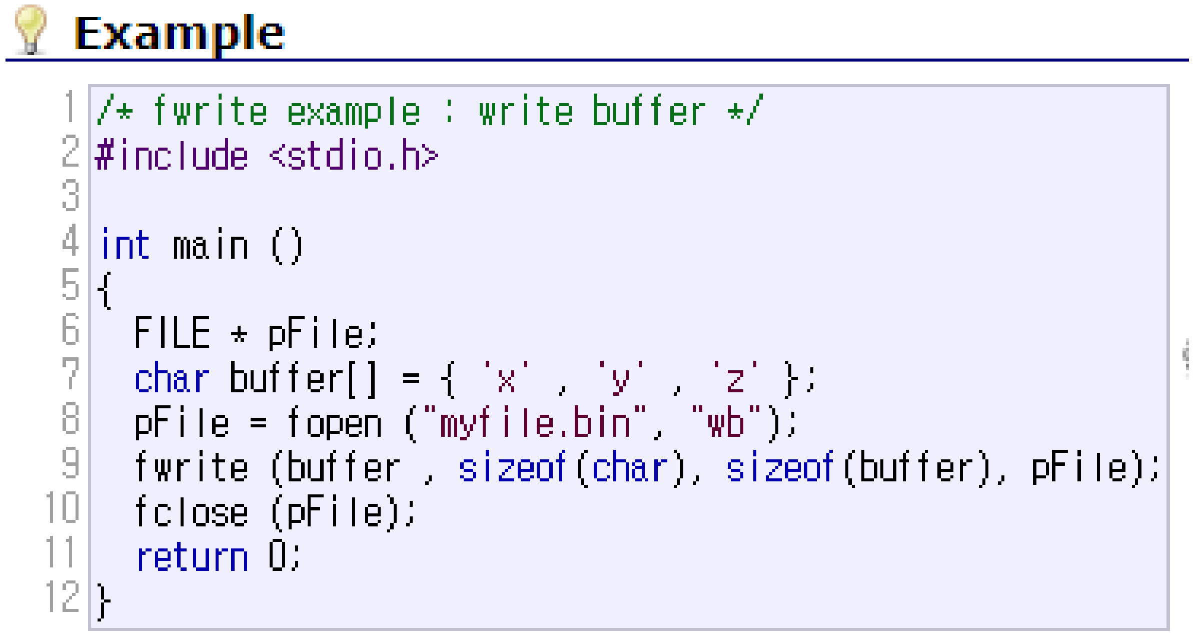


Example

```
/* fopen example */
#include <stdio.h>
int main ()
{
  FILE * pFile;
  pFile = fopen ("myfile.txt","w");
  if (pFile!=NULL)
  {
    fputs ("fopen example",pFile);
    fclose (pFile);
  }
  return 0;
}
```

C++ I/O library

- size_t **fwrite** (const void * ptr, size_t size, size_t count, FILE * stream);

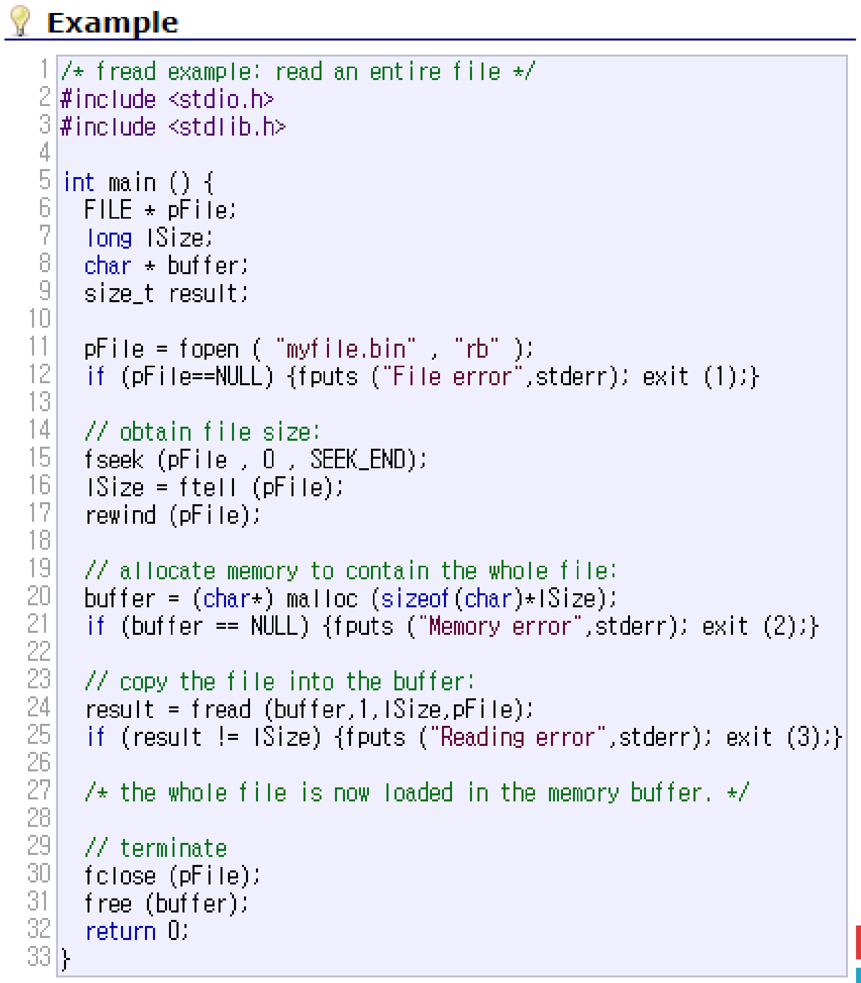


Example

```c
/* fwrite example : write buffer */
#include <stdio.h>

int main ()
{
  FILE * pFile;
  char buffer[] = { 'x' , 'y' , 'z' };
  pFile = fopen ("myfile.bin", "wb");
  fwrite (buffer , sizeof(char), sizeof(buffer), pFile);
  fclose (pFile);
  return 0;
}
```

C++ I/O library

- size_t **fread** (void * ptr, size_t size, size_t count, FILE * stream);

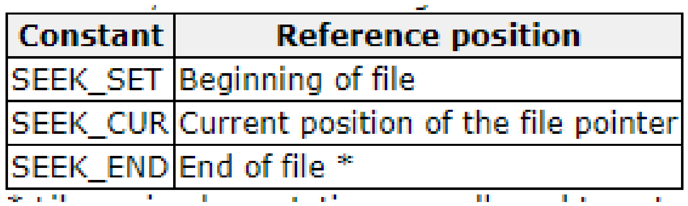


Example

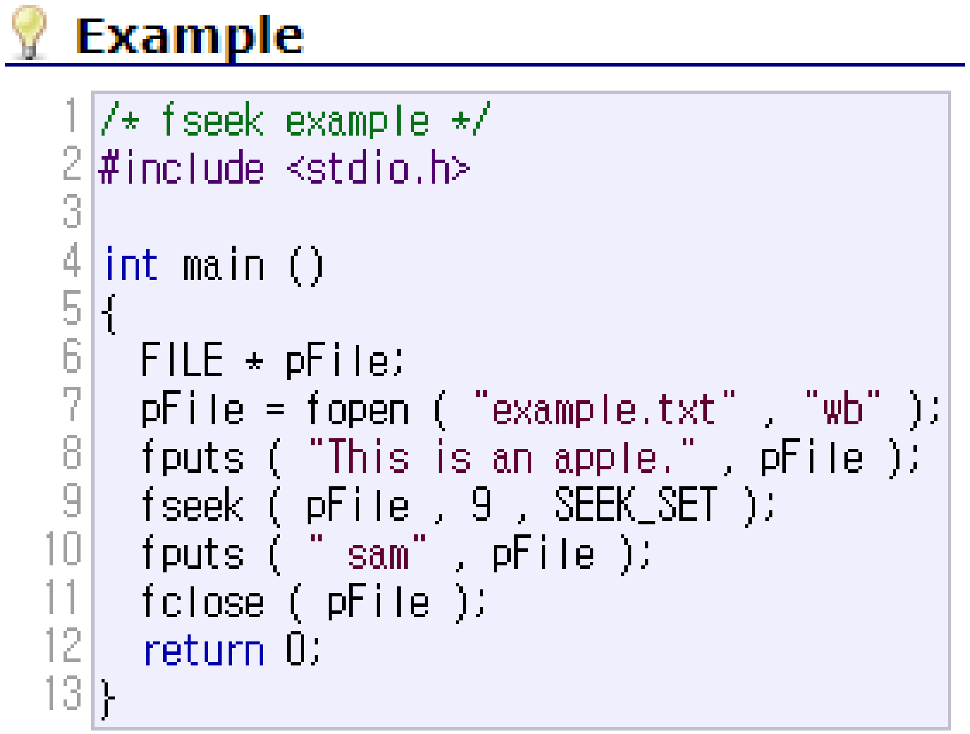
```
/* fread example: read an entire file */
#include <stdio.h>
#include <stdlib.h>

int main () {
  FILE * pFile;
  long lSize;
  char * buffer;
  size_t result;

  pFile = fopen ( "myfile.bin" , "rb" );
  if (pFile==NULL) {fputs ("File error",stderr); exit (1);}

  // obtain file size:
  fseek (pFile , 0 , SEEK_END);
  lSize = ftell (pFile);
  rewind (pFile);

  // allocate memory to contain the whole file:
  buffer = (char*) malloc (sizeof(char)*lSize);
  if (buffer == NULL) {fputs ("Memory error",stderr); exit (2);}

  // copy the file into the buffer:
  result = fread (buffer,1,lSize,pFile);
  if (result != lSize) {fputs ("Reading error",stderr); exit (3);}

  /* the whole file is now loaded in the memory buffer. */

  // terminate
  fclose (pFile);
  free (buffer);
  return 0;
}
```

C++ I/O library

- int **fseek** (FILE * stream, long int offset, int origin);



Constant	Reference position
SEEK_SET	Beginning of file
SEEK_CUR	Current position of the file pointer
SEEK_END	End of file *



Example

```
/* fseek example */
#include <stdio.h>

int main ()
{
  FILE * pFile;
  pFile = fopen ( "example.txt" , "wb" );
  fputs ( "This is an apple." , pFile );
  fseek ( pFile , 9 , SEEK_SET );
  fputs ( " sam" , pFile );
  fclose ( pFile );
  return 0;
}
```