FILES

Dr Frank Guan

INF1002 – Programming Fundamentals

Week 12

1. Files (sample code uploaded to LMS)
2. Recap of all past lectures

FILES | SiT SINGAPORE INSTITUTE OF TECHNOLOGY

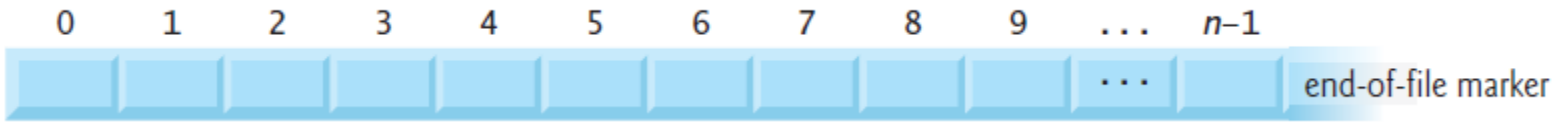
INTRODUCTION

Storage of data in variables and arrays is temporary—such data is lost when a program terminates.

Files are used for permanent retention of data. Computers store files on secondary storage devices such as hard disks and flash memory.

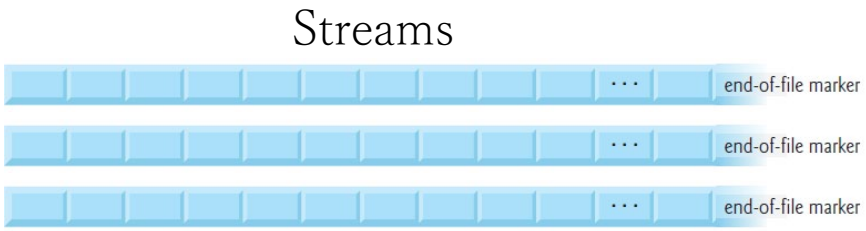
FILES & STREAMS

C views each file as a sequential stream of bytes. Each file ends at the **end-of-file** (EOF) **marker**.



STREAMS

Files



Programs

When a file is opened, a `stream` is associated with the file

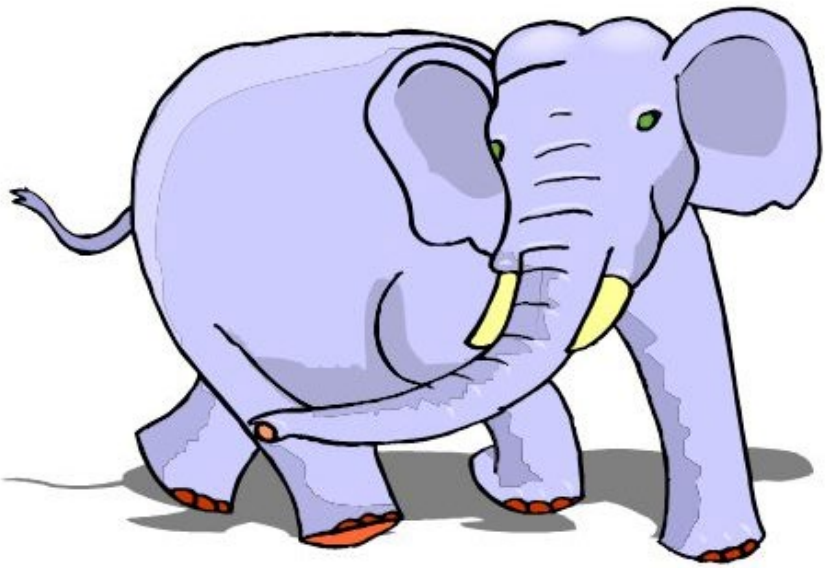
BASIC FILE OPERATIONS

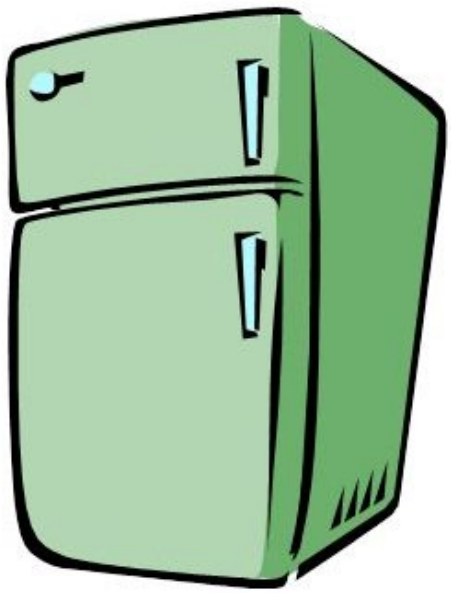
- Create a new file and write content into it
- Open an existing file and read content from it
- Basic functions:
  - fopen
  - fprintf
  - fscanf
  - feof
  - fclose
  - fseek
  - fread
  - fwrite
  - …

CREATING AND WRITING INTO A TEXT FILE

Question:

How do you put an elephant into a refrigerator?





SAMPLE: CREATING AND WRITING A TEXT FILE

create_file.c

```
#include <stdio.h>

int main() {

        /* declare a pointer to a FILE structure */
        FILE *f;

        /* open the file with fopen() */
        f = fopen("create_file.txt", "w");
        if (f == NULL) {
                printf("Could not open data.txt.\n");
                return 1;
        }

        /* write to the file with fprintf() */
        fprintf(f, "Hello! This is a new file.\n");

        /* close the file with fclose() */
        fclose(f);

        return 0;

}
```

Step 0: declare a pointer to a FILE structure

Step 1: Open the file using fopen()

Step 2: Write to the file using fprintf()

Step 3: Close the file using fclose()

"FILE" STRUCTURE

- A **`structure`** data type
- Contains info to control a stream
- The content info is not meant to be accessed from outside the functions of the <stdio.h> and <cwchar.h>

Only for your reference:

```
typedef struct _iobuf
{
    char*   _ptr;
    int _cnt;
    char*   _base;
    int _flag;
    int _file;
    int _charbuf;
    int _bufsiz;
    char*   _tmpfname;
} FILE;
```

From stdio.h in MinGW32 5.1.4

STEP 1: OPEN (AND CREATE) A TEXT FILE

```
f = fopen("data.txt", "w");
```

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

Opens the file whose name is specified in the parameter *filename* and associates it with a stream. The operations that are allowed on the stream and how these are performed are defined by the *mode* parameter.

File opening modes for text files:

mode	Description
"r"	Open a file for reading. The file must exist.
"w"	Create an empty file for writing. If a file with the same name already exists its content is erased and the file is considered as a new empty file.
"a"	Append to a file. Writing operations 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 exist.
"w+"	Create an empty file for both reading and writing.
"a+"	Open a file for reading and appending.

STEP 2: WRITE TO A TEXT FILE

```
fprintf(f, "Hello! This is a new file.\n");
```

```
int fprintf ( FILE * stream, const char * format, ... );
```

Writes the C string pointed by format to the *stream*. If *format* includes format specifiers (subsequences beginning with %), the *additional arguments* following format are formatted and inserted in the resulting string replacing their respective specifiers.

STEP 3: CLOSE A TEXT FILE

```
fclose(f);
```

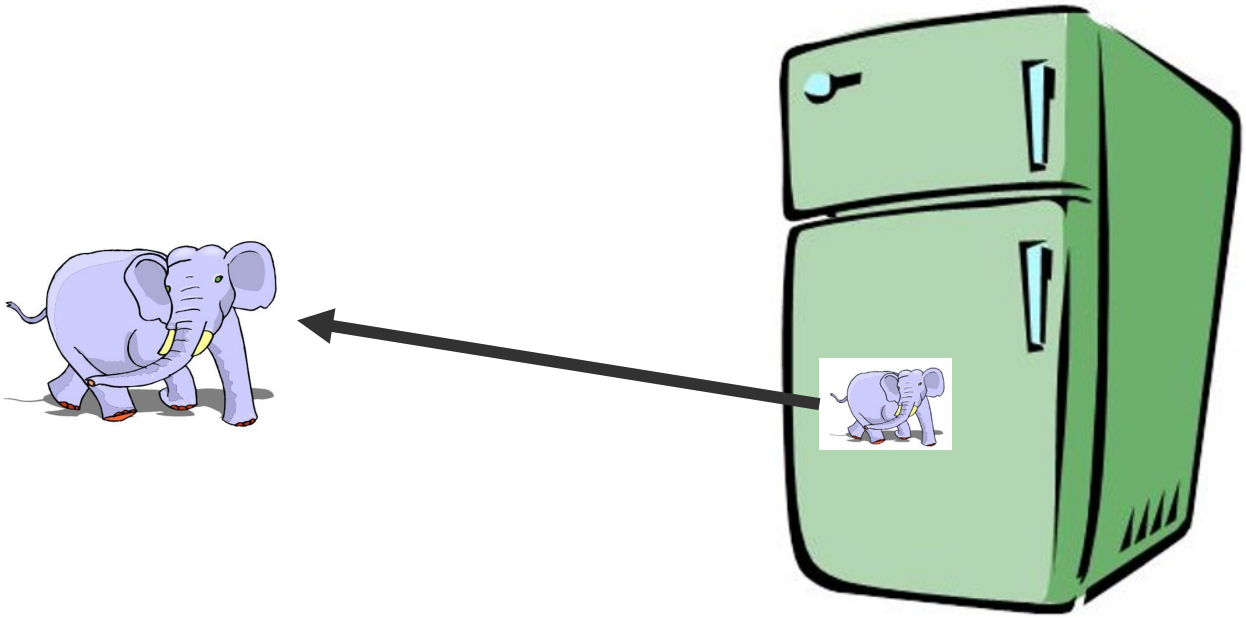
```
int fclose ( FILE * stream );
```

Closes the file associated with the *stream* and disassociates it.

READING FROM A TEXT FILE

Question:

How do you take out an elephant from the refrigerator?



READ DATA FROM A TEXT FILE

```
int fscanf ( FILE * stream, const char * format, ... );
```

Reads data from the *stream* and stores them according to the parameter *format* into the locations pointed by the additional arguments.

```
fscanf  (f, "%d%19s%lf", &account, name, &balance);
```

Account	Name	Balance
10000	Frank	100.00
20000	Daniel	200.00

READING FROM A TEXT FILE - EXAMPLE

```
#include <stdio.h>
int main() {
        FILE *f;
        int account;
        char name[20];
        double balance;
        /* open the file */
        f = fopen("read_from_file.txt", "r");
        if (f == NULL) {
                printf("Could not open credit.txt.\n");
                return 1;
        }
        /* read until the end of the file */
        printf("%10s\t%20s\t%10s\n", "Account", "Name", "Balance");
        while (!feof(f)) {
                /* read one record */
                fscanf(f, "%d%19s%lf", &account, name, &balance);
                /* display it to the screen */
                printf("%10d\t%20s\t%10.2lf\n", account, name, balance);
        }
        /* clean up */
        fclose(f);
        return 0;
}
```

Step 0: declare a pointer to a FILE structure

Step 1: Open the file using fopen()

Step 2: Read data from the file using fscanf()

`feof()` returns a true value if the file pointer is at the end of the file.

Step 3: Close the file using fclose()

read_from_file.c

QUESTION

```c
#include <stdio.h>
int main() {
        FILE *f;
        int account;
        char name[20];
        double balance;
        /* open the file */
        f = fopen("read_from_file.txt", "r");
        if (f == NULL) {
                printf("Could not open credit.txt.\n");
                return 1;
        }
        /* read until the end of the file */
        printf("%10s\t%20s\t%10s\n", "Account", "Name", "Balance");
        while (!feof(f)) {
                /* read one record */
                fscanf(f, "%d%19s%lf", &account, name, &balance);
                /* display it to the screen */
                printf("%10d\t%20s\t%10.2lf\n", account, name, balance);
        }
        /* clean up */
        fclose(f);
        return 0;
}
```

read_from_file.c

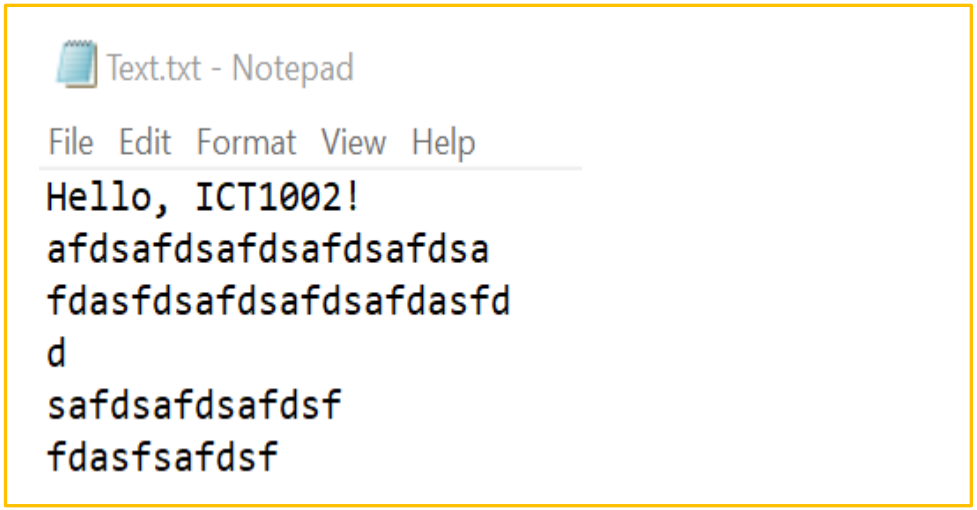
The above code will go through all content from the beginning until reaching the EOF.

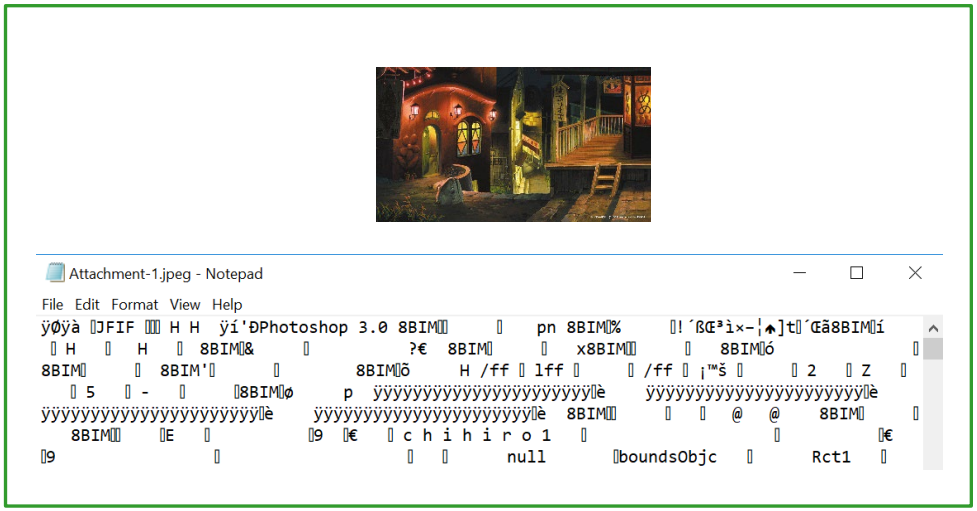
What if we want to read/write data at a random location inside a file?

TWO STREAM OPENING MODES

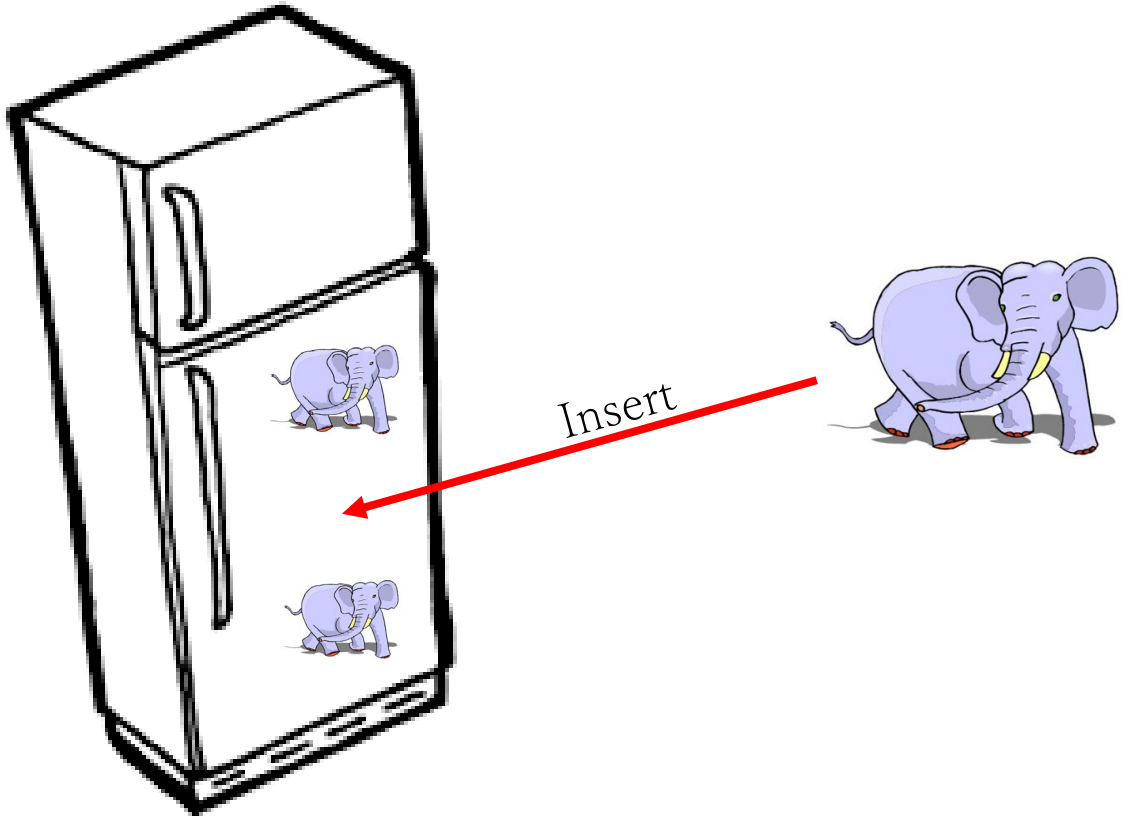
Streams may be opened in one of two modes:

- `text mode` is used for reading and writing text files
  - files are divided into lines by new line characters
  - e.g. plain text, source code, HTML, XML, etc.
- `binary mode` is used for reading and writing all other files
  - files are made up of raw bytes
  - e.g. images, videos, databases, etc.





There are two elephants in the refrigerator now. How can I insert a new elephant between the two elephants???



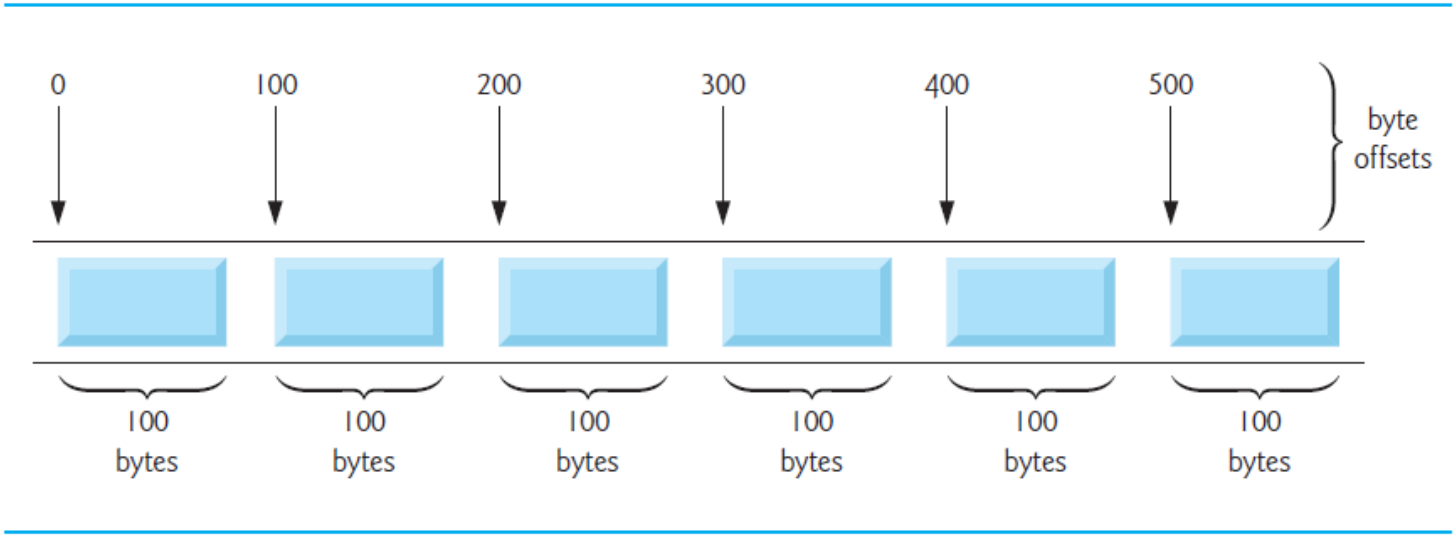
RANDOM ACCESS FILES

A random access file can be read or written in any order.

Writing to one part of the file does not change another part of the file.

RANDOM ACCESS FILES

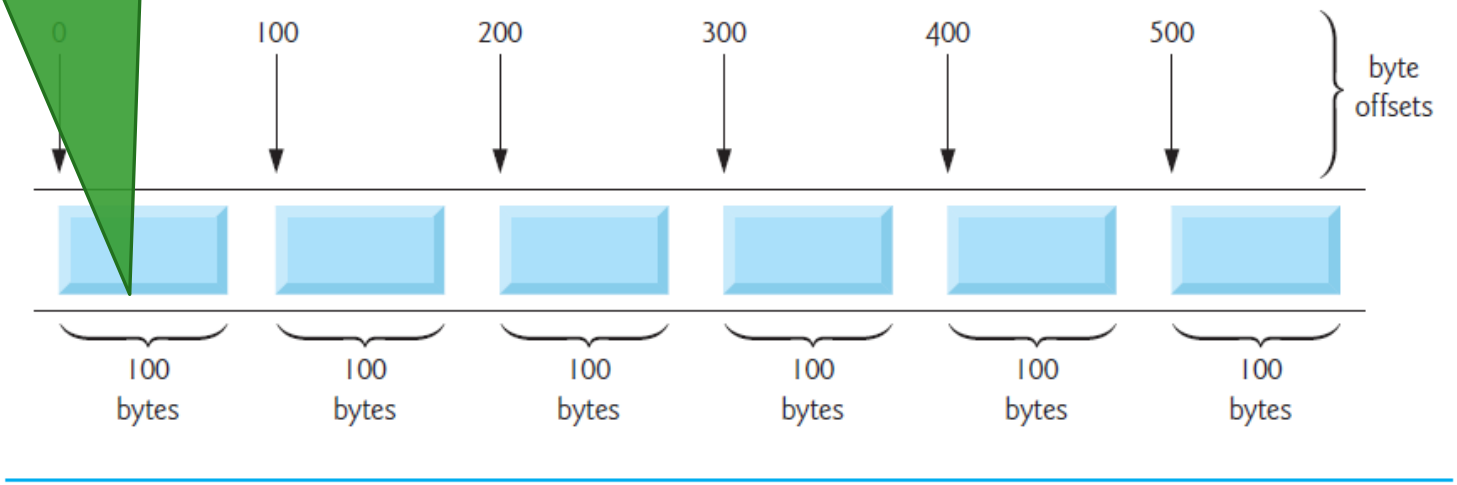
The simplest kind of random access file consists of a series of records of fixed length:



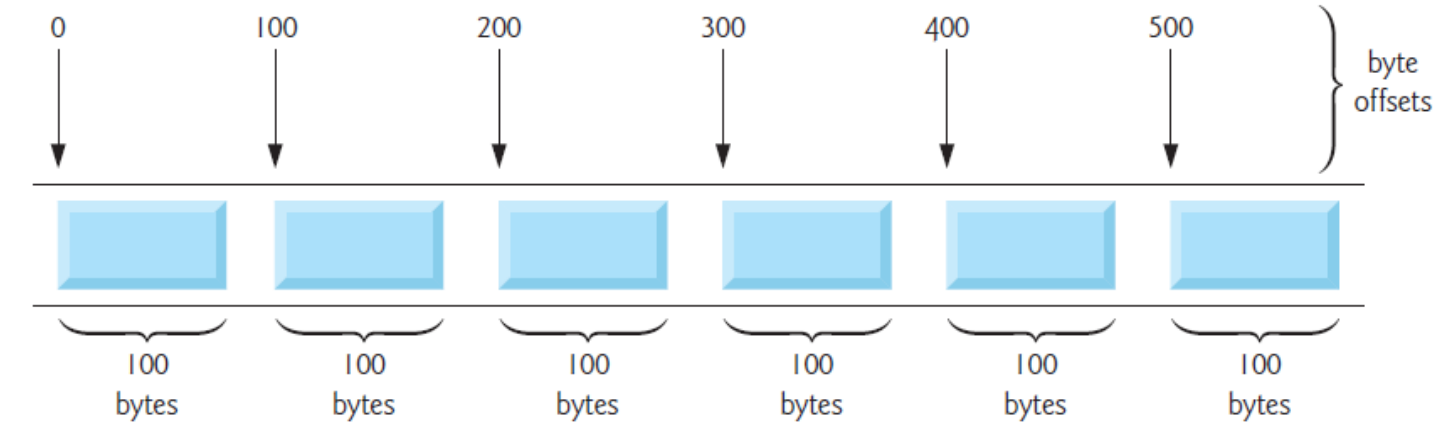
RANDOM ACCESS FILES

Every record in a random access file has the same length

The exact location of a record can be calculated from the record key and length.



RANDOM ACCESS FILES



Fixed-length records **enable data to be inserted/updated/deleted** in a random access file **without destroying** other data

STEP 1: OPENING A RANDOM ACCESS FILE

```
f = fopen(filename, “wb+");
```

```
f = fopen(filename, "wb+");
```

Binary file opening modes:

rb	Opens an existing binary file for reading.
wb	Opens a binary file for writing. If it does not exist, a new file is created. If it exists, it will be overwritten.
ab	Opens a binary file for appending. If it does not exist, a new file is created.
rb+	Opens a binary file for reading and writing both.
wb+	Opens a binary file for reading and writing both.
ab+	Opens a binary file for reading and writing both.

STEP 2: USE `fseek` TO MOVE TO A SPECIFIC POSITION IN THE FILE

```
fseek(f, (client.acc_num - 1) * sizeof(Client), SEEK_SET);
```

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

Sets the position indicator associated with the *stream* to a new position.

For streams open in binary mode, the new position is defined by adding *offset* to a reference position specified by *origin*.

origin

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

STEP 3: USE FWRITE/FREAD TO WRITE/READ

fwrite(&client, sizeof(Client), 1, f);

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

Writes an array of *count* elements, each one with a size of *size* bytes, from the block of memory pointed by *ptr* to the current position in the *stream*.

fread(&client, sizeof(Client), 1, f);

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

Reads an array of *count* elements, each one with a size of *size* bytes, from the *stream* and stores them in the block of memory specified by *ptr*.

SAMPLE: READ FROM RANDOM ACCESS FILE

```
/*
 * Random access file example.
 */
#include <stdio.h>

/* this structure holds the data for one client */
typedef struct client_struct {
    int acc_num;
    char last_name[15];
    char first_name[10];
    double balance;
} Client;

int main() {
        const char *filename = "random_access_write.dat";
        FILE *f;
        Client client;

        /* open the data file */
        f = fopen(filename, "rb");
        if (f == NULL) {
                printf("Could not open %s.\n", filename);
                return;
        }
        /* print title */
        printf("%-6s%-16s%-11s%10s\n", "Acct", "Last Name", "First Name", "Balance");
        /* read one record at a time until we reach EOF */
        fread(&client, sizeof(Client), 1, f);
        while (!feof(f)) {
                if (client.acc_num != 0)
                        printf("%-6d%-16s%-11s%10.2lf\n", client.acc_num, client.last_name,
client.first_name, client.balance);
                fread(&client, sizeof(Client), 1, f);
        }
        fclose(f);
}
```

Step 0: declare a pointer to a FILE structure

Step 1: Open the file using fopen() in binary mode

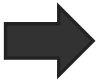
Step 2: Read the data from the present position in the file

Step 3: Close the file using fclose()

SAMPLE: READ FROM RANDOM ACCESS FILE

- random_access_read.c

```
Enter account number (1-100, 0 to end)
? 1
Enter last_name first_name balance
? Guan Frank 100
Enter account number (1-100, 0 to end)
? 2
Enter last_name first_name balance
? Wang Daniel 200
Enter account number (1-100, 0 to end)
? 0

Process returned 0 (0x0)   execution time : 18.544 s
Press any key to continue.
```



```
Acct  Last Name     First Name    Balance
1     Guan          Frank          100.00
2     Wang          Daniel         200.00

Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.
```

SAMPLE: WRITE TO RANDOM ACCESS FILE

random_access_write.c

```
#include <stdio.h>
/* this structure holds the data for one client */
typedef struct client_struct {
    int acc_num;
    char last_name[15];
    char first_name[10];
    double balance;
} Client;

int main() {
        const char *filename = "random_access_write.dat";
        FILE *f;
        Client client;
        /* open the data file */
        f = fopen(filename, "wb+");
        if (f == NULL) {
                printf("Could not open %s.\n", filename);
                return;
        }
        /* read account data from the user */
        printf("Enter account number (1-100, 0 to end)\n? ");
        scanf("%d", &client.acc_num);
        while (client.acc_num != 0) {
                /* read the data for this record */
                printf("Enter last name, first name and balance\n? ");
                scanf("%14s%9s%lf", client.last_name, client.first_name, &client.balance);
                /* go to this record's position in the file */
                fseek(f, (client.acc_num - 1) * sizeof(Client), SEEK_SET);
                /* write the client data structure */
                fwrite(&client, sizeof(Client), 1, f);
                /* ask for another record */
                printf("Enter account number (1-100, 0 to end)\n? ");
                scanf("%d", &client.acc_num);
        }
        fclose(f);
        return 0;
}
```

Step 0: declare a pointer to a FILE structure

Step 1: Open the file using fopen() in binary mode

Step 2: Move to a certain position of the file

Step 3: Write to the file using fwrite()

Step 4: Close the file using fclose()

SAMPLE: WRITE TO RANDOM ACCESS FILE

– “random_access_write.c

```
Enter account number (1-100, 0 to end)
? 1
Enter last_name first_name balance
? Guan Frank 100
Enter account number (1-100, 0 to end)
? 2
Enter last_name first_name balance
? Wang Daniel 200
Enter account number (1-100, 0 to end)
? 0

Process returned 0 (0x0)   execution time : 18.544 s
Press any key to continue.
```

END-OF-WEEK CHECKLIST

- [ ] Files
- [ ] Streams
- [ ] Opening a file
- [ ] Writing to a file
- [ ] Reading from a file
- [ ] Sequential access files
- [ ] Random access files

TAKE A LOOK BACK

SINGAPORE INSTITUTE OF TECHNOLOGY

A SIMPLE C PROGRAM

```
/*
 * Hello World program in C.
 */
#include <stdio.h>

int main() {

        printf("Hello world!\n");

        return 0;

}
```

```
/*
 * Hello World program in C.
 */
#include <stdio.h>

int main() {

        printf("Hello world!\n");

        return 0;

}
```

C programs contain one or more functions, one of which MUST be `main`. Every program in C begins execution in `main`.

#DEFINE/#INCLUDE PRE-PROCESSOR DIRECTIVE

The #include directive causes a copy of a specified file to be included in place of the directive.

```
#include <stdio.h>
#define NUM_STUDENTS 140

int main() {

    int scores[NUM_STUDENTS];

    for (int i = 0; i < NUM_STUDENTS; i++) {
        scores[i] = 0;
    }

    return 0;
}
```

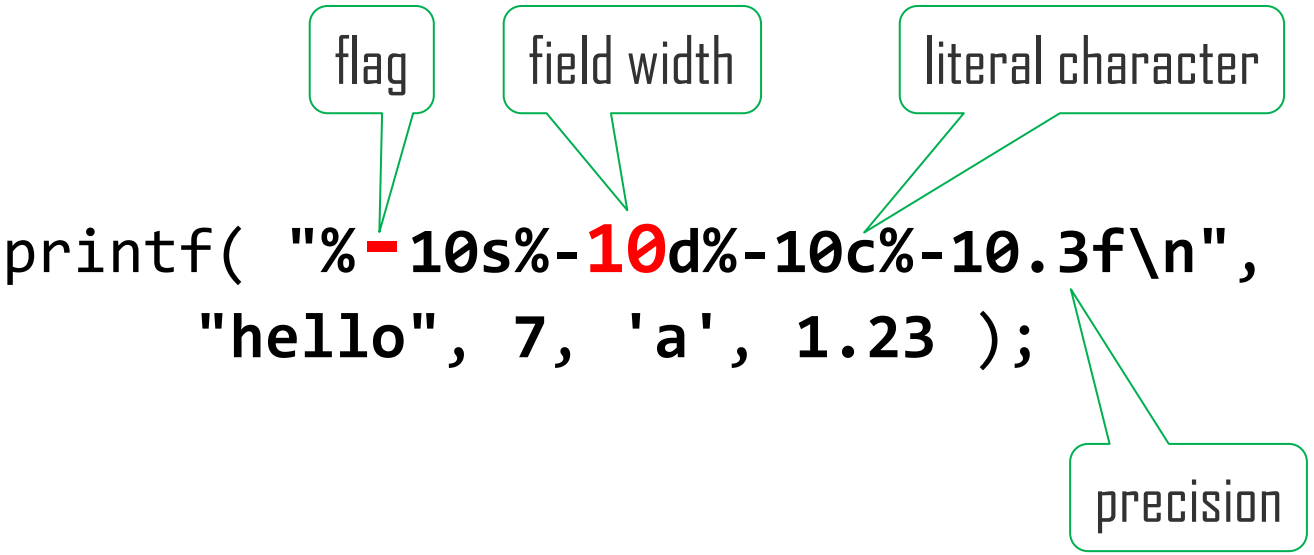
The #define directive creates symbolic constants.

All subsequent occurrences of NUM_STUDENTS will be replaced with 140.

```
#include <stdio.h>
#define NUM_STUDENTS 140

int main() {

    int scores[NUM_STUDENTS];

    for (int i = 0; i < NUM_STUDENTS; i++) {
        scores[i] = 0;
    }

    return 0;
}
```

PRINTF

conversion-specifier = <flags><field width><precision><literal character>



SCANF – READING FORMATTED INPUT

```
scanf(format-control-string, other-arguments);
```

```
printf("Enter seven integers: ");
scanf("%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g);

printf("The input displayed as decimal integers is:\n");
printf("%d %d %d %d %d %d %d", a, b, c, d, e, f, g);
```

Output

```
Enter seven integers: -70 -70 070 0x70 70 70 70
The input displayed as decimal integers is:
-70 -70 56 112 56 70 112
```

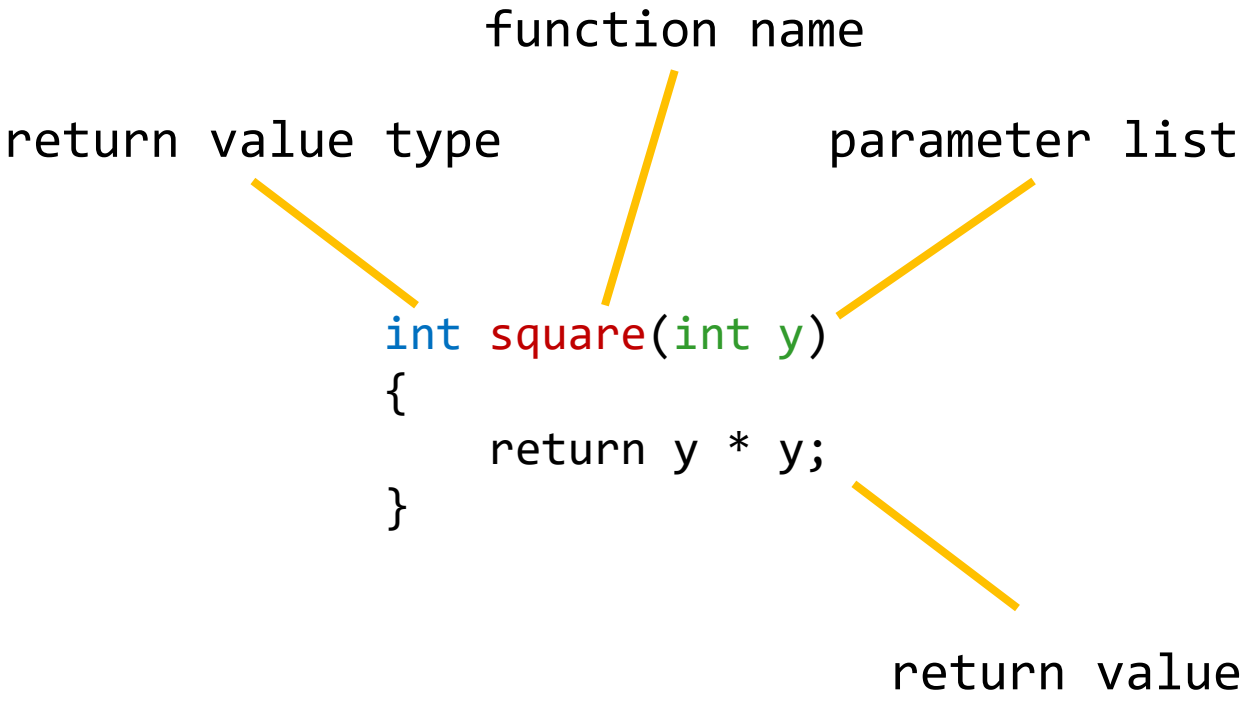
C CONTROL STRUCTURES

C has the same control structures as other programming languages:

- `if-else`
- `switch`
- `for`
- `while`
- `do-while`

FUNCTION

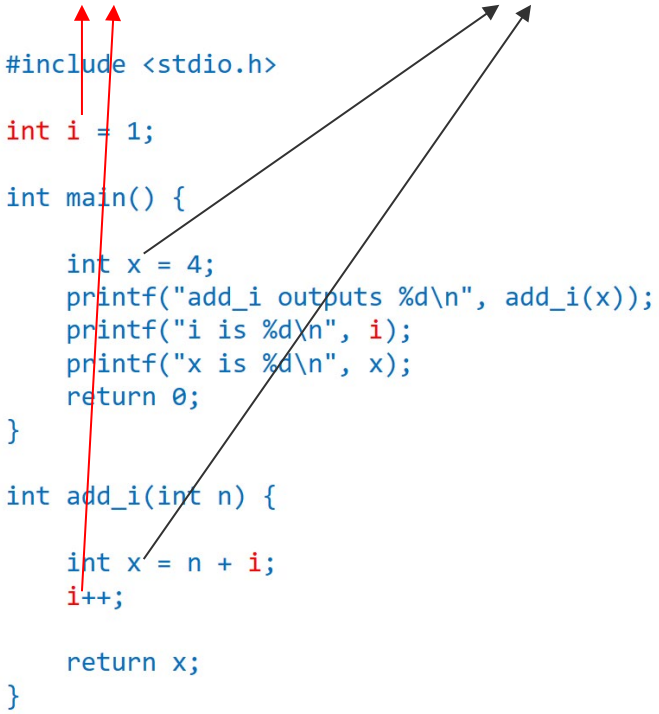
- Function



- A **function prototype** is a function definition without a body, e.g.
  - `int square(int);`

SCOPE

- The **scope** of an identifier is the portion of the program in which the identifier can be referenced.
- Global VS Local variable

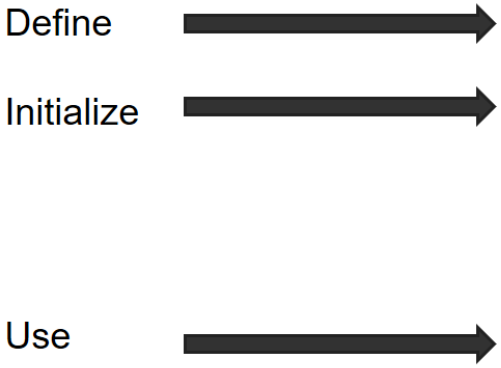


```c
#include <stdio.h>

int i = 1;

int main() {

    int x = 4;
    printf("add_i outputs %d\n", add_i(x));
    printf("i is %d\n", i);
    printf("x is %d\n", x);
    return 0;
}

int add_i(int n) {

    int x = n + i;
    i++;

    return x;
}
```

ARRAY

An array is a group of memory locations that all have

- the same **name**
- the same **type**

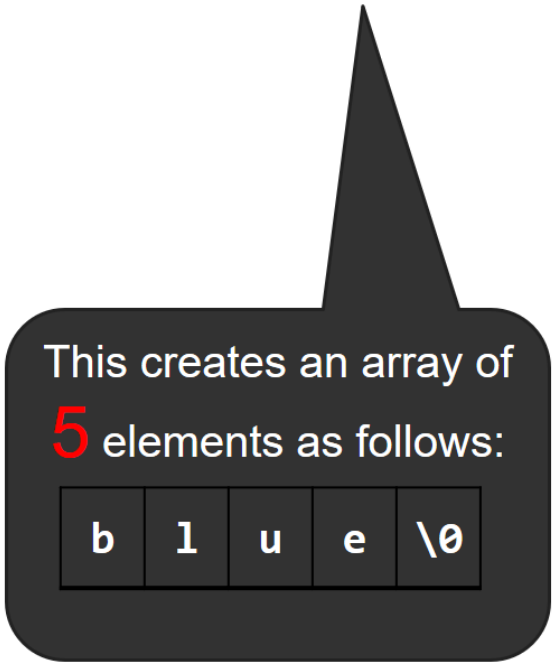



```c
#include <stdio.h>

#define MAX_STUDENTS 10

int main() {

    int studentId[MAX_STUDENTS];

    for (int i = 0; i < MAX_STUDENTS; i++)
        studentId[i] = i + 1;

    printf("%7s%13s\n", "Element", "Value");

    for (int i = 0; i < MAX_STUDENTS; i++)
        printf("%7d%13d\n", i, studentId[i]);

    return 0;
}
```

STRING

A string in C is an **array** of characters **ending in the null character** (**`'\0'`**).

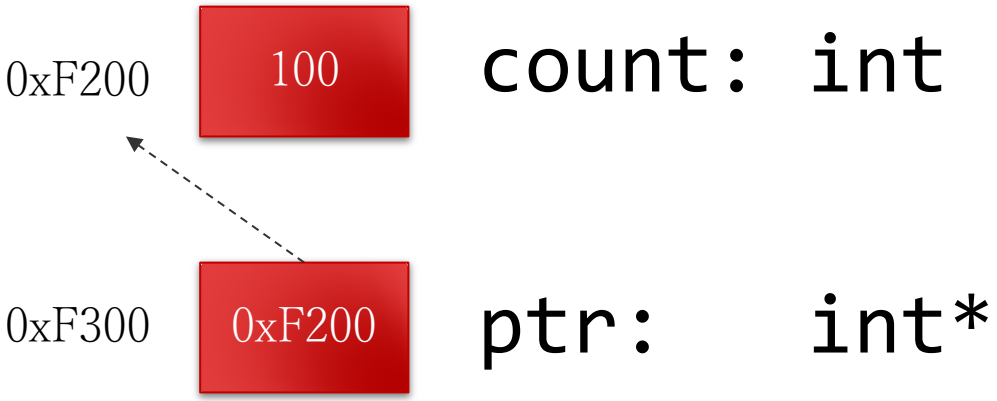
```
char colour[] = "blue";
```



POINTER

Pointers are variables whose values are memory addresses.

```
int count = 100;

int* ptr = &count;
```



HOW TO USE POINTER

- D: Declaration/definition
  - int variable;
  - int *ptr;
- I: Initialization (value assignment)
  - int variable = 10;
  - ptr = &variable;
- D: Dereference
  - *ptr = 20; (update the value stored in the pointed memory space)
  - int a = *ptr; (retrieve the value stored in the pointed memory space)

POINTERS AND ARRAY

Pointers and arrays are intimately related in C.

- An array name can be thought of as a `constant pointer` to the start of the array.
- Array subscripts can be applied to pointers.
- Pointer arithmetic can be used to navigate arrays.

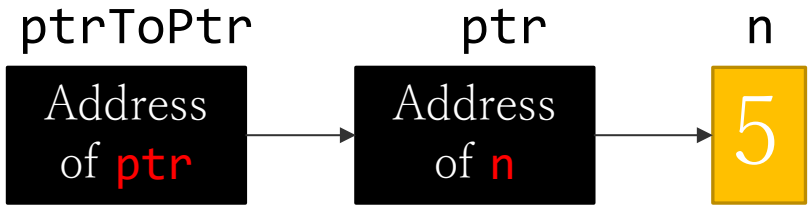
```
int main() {

    char b[] = {'a', 'b', 'c', 'd', 'e' };
    char *bPtr = b;

    printf("*(bPtr + 3): \t%c\n", *(bPtr + 3));
    printf("*(b + 3): \t%c\n", *(b + 3));
    printf("bPtr[3]: \t%c\n", bPtr[3]);

        return 0;
}
```

Output

```
*(bPtr + 3): d
*(b + 3):    d
bPtr[3]:     d
```

POINTER TO POINTER

```
int n = 5;
int *ptr = &n;
int **ptrToPtr = &ptr;
```



`(int *)`: pointer to an integer

`(int *)*`: pointer to a pointer which points to an integer

Many uses in C:

- Arrays of pointers
- Arrays of strings

CALL BY VALUE VS CALL BY REFERENCE

Call by Reference

pass by reference

cup =

fillCup( )

Call by Value

pass by value

cup =

fillCup( )

www.mathwarehouse.com

STRUCTURES

C allows structured collections of information to be defined using the struct keyword.

```
struct <name> {
   member 1;
   member 2;
   :
   member n;
};
```

DYNAMIC MEMORY ALLOCATION

- Three steps to dynamic memory allocation:

1. include

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

2. malloc

Use `malloc` or `calloc` to request memory.

```
int *ptr = (int *)malloc(sizeof(int)*N);
```

3. free

Free up the memory when no longer needed.

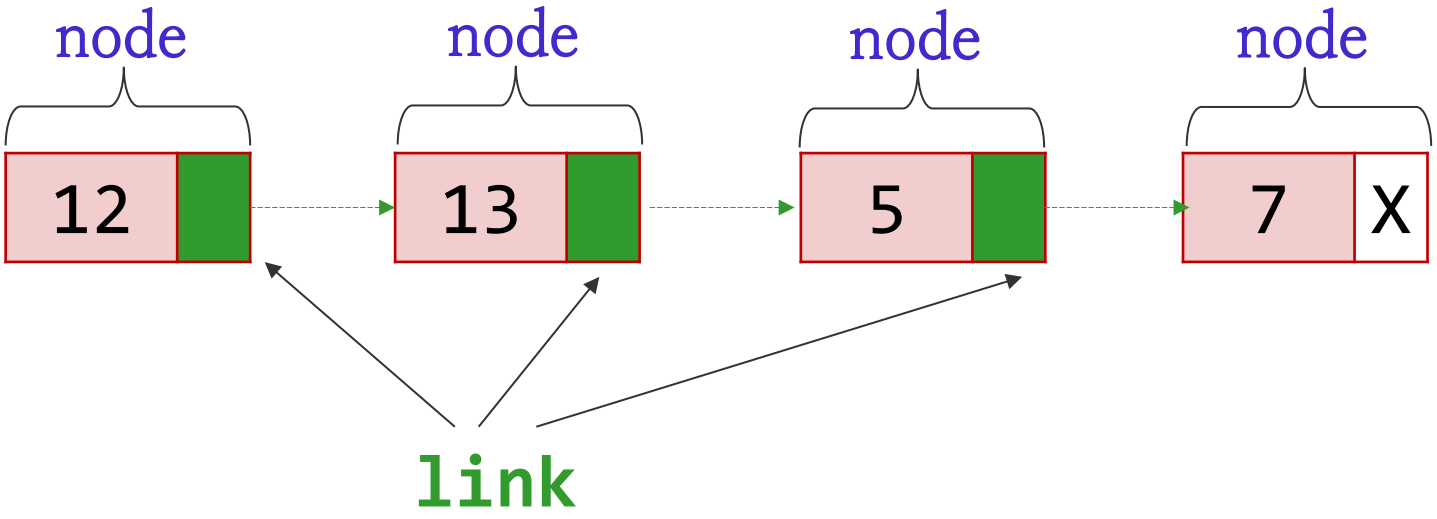
```
free(ptr);
```

FREE ALLOCATED MEMORY

`free` de-allocates memory previously allocated by `malloc` or `calloc`

- all memory allocated by `malloc` or `calloc` MUST be `free`'d
- this allows the memory to be re-used
- failure to de-allocate memory is called a memory leak
- a leaking program will use up more and more memory over time, and eventually crash

LINKED LIST



A linked list is a linear collection of self-referential structures, called nodes, connected by pointers, called links.

How do you create two nodes and link **node1** to **node2**?

```
int main() {

        Node node1 = { 15, NULL };
        Node node2 = { 10, NULL };
        node1.next = &node2;

}
```

node1 node2

LINKED LIST OPERATIONS

Search/update

Insert

Delete