```
#include <stdio.h> <stdlib.h> <string.h> <stdbool.h> <math.h> <ctype.h> <limits.h>
```

```
# FILE & DIRECTORY
ls -la          # List with all info
cd /path        # Change directory
pwd             # Show current directory
mkdir name      # Make directory
rmdir name      # Remove empty directory
rm file | -r dir # Remove file/dir
cp src dest     # Copy file/dir
mv src dest     # Move/rename
touch file      # Create file
find / -name "f" # Find file
# VIEW & EDIT FILES
cat file        # Show file content
more/less file  # View file paged
nano/vi/vim file # Edit file
head -n 10 file  # First 10 lines
tail -f file    # Last lines (follow)
# PERM
chmod 7
chown u
# SYST
uname -
df -h
du -sh d
top / hto
free -h
uptime
who
# PROC
ps aux
kill PID
killall na
jobs
bg / fg
# NETW
ping hos
curl URL
wget UR
ss / nets
# COMP
tar -cvf f
gzip file
gunzip fi
zip f.zip
unzip f.z
# MISC
echo "te
date
man cm
alias ll="
history
clear
# GITH
cd path/
git init
git add
git comm
# GDB -
break m
break 10
break fil
delete
delete 1
enable 1
info brea
# GDB -
run
continue
breakpo
next / n
step / s
finish / f
# GDB -
info loca
print x
print arr
set var x
watch x
# GDB -
backtrac
stack)
frame 1 /
# GDB -
list / l
layout sr
refresh
# Valgr
gcc -g fi
valgrind
valgrind
# gprof
gcc -pg
./a.out
gprof ./a
```

```
#FILEIO
FILE *in
FILE *ou
double x
while (fs
    fprint
    if (fge
}
fclose(in
FILE* f =
f = fopen
f = fopen("file.txt", "a");    // append mode

fprintf(f, "x=%d\n", x);       // write to file
fscanf(f, "%lf", &y);          // read a float
fgetc(f), fputc('A', f);       // char I/O
fclose(f);                     // close file
```

```
#STRING FUNCTIONS (string.h)
strcpy(dest, src)       // copy string
strlen(str)             // string length
strcmp(s1, s2)          // compare strings
strcat(dest, src)       // append
strchr(str, 'c')        // find char in string
```

```
# ASCII
'0' 48   '9' 57     'A' 65   'Z' 90     'a' 97   'z' 122
# Integer Types
char   1 B  %c    [-128 to 127]
short  2 B  %hd  [-32K to 32K]
int    4 B  %d/%i   [-2.1B to 2.1B]
long   8 B   %ld   [-9.2Q to 9.2Q]
long long  8 B   %lld    same as long
# Unsigned Integer Types
size_t %zu
unsigned char  1 B     %c  [0 to 255]
unsigned short   2 B     %hu [0 to 65K]
unsigned int   4 B     %u  [0 to 4.2B]
unsigned long     8 B     %lu  [0 to 18.4Q]
unsigned long long 8 B     %llu  same as above
# Floating Point Types
float       4 B     %f       ~6-7 decimal digits
```

```
void freeHashTable() {
    for (int i = 0; i < TABLE_SIZE; i++) {
        Hashentry* temp = hashtable[i];  // head of list
        while (temp) {
            Hashentry* tofree = temp;  // store node to free
            temp = temp->next;    // move to next
            free(tofree);  // free node
        }
    }
}
int main() {
    insert(1, 100);  // insert key 1
    insert(15, 1200);   // insert key 15 (same bucket as 1)
    insert(25, 2200);    // insert key 25 (same bucket)
    printf("Value for key 1: %d\n", search(1));  // print value for 1
    printf("Value for key 15: %d\n", search(15)); // print value for 15
    printf("Value for key 25: %d\n", search(25));  // print value for 25
    freeHashTable();  // cleanup
    return 0;
}
```

```
# STRUCT
typedef struct ListNode {
    int data;
    struct ListNode* next;
}ListNode;
# TRAVERSE LIST
for (ListNode* cur = head; cur != NULL;
cur = cur->next) {
    printf("ListNode: %d, addr: %p\n",
cur->data, cur); }
# ADD TO HEAD
ListNode* addToHead(...) {
    node->next = head;
    return node; }
# APPEND TO TAIL
ListNode* appendToTail(...) {
    node->next = NULL;
```

```
sizeof(int));
            if (!arr[i][j][k]) return NULL;   }}}
    return arr;}
void free4Darray(int ****arr, int d1, int d2,
int d3) {
    for (int i = 0; i < d1; i++) {
        for (int j = 0; j < d2; j++) {
            for (int k = 0; k < d3; k++) {
                free(arr[i][j][k]);}
            free(arr[i][j]);}
        free(arr[i]);}
    free(arr);}
```

```
# SINGLY LINKED
// Node structure
typedef struct Node {
    int data;
    struct Node* next;
} Node;
// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
```

```
    }
Node* copyList(Node* head) {
    if (head == NULL) return NULL; // If list is empty

    Node* newHead = createNode(head->data); // First node copy
    Node* currOld = head->next;
    Node* currNew = newHead;

    while (currOld != NULL) { // Loop through old list
        currNew->next = createNode(currOld->data); // Create + link node
        currNew = currNew->next; // Move forward in new list
        currOld = currOld->next; // Move forward in old list
    }
    return newHead; // Return head of copied list
}
```

```
#DOUBLY LINKED
// Node structure
typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;
// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}
// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
```

```
    Node* temp = *head;
    Node* prev = NULL;

    if (temp != NULL && temp->data == key) { // Deleting head
        *head = temp->next;
        free(temp);
        return;
    }

    while (temp != NULL && temp->data != key) { // Search node
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) return; // Not found

    prev->next = temp->next; // Remove link
    free(temp); // Free memory
}
```