Fibonacci Series

A series of numbers in which each number (Fibonacci number) is the sum of the two preceding numbers

The simplest is the series 1, 1, 2, 3, 5, 8, etc.

Fibonacci Series Program

```
#include<iostream>
using namespace std;



int main()
{

	int num;
	int first=0,second=1,next;

	cout<<endl<<"Enter the Series Element No:::";
	cin>>num;

	for(int i=0;i<num;i++)
	{
		if(i<=1)
		{
			next=i;
		}
		else
		{
			next=first+second;
			first=second;
			second=next;

		}
		cout<<endl<<next<<"\t";
	}

	return 0;
}
```

Square Root Program

```
#include<iostream>
using namespace std;

int sqrt32(int number)
{
```

```
        int square=1;
        int delta=3;

        while( square <=number)
        {
                square +=delta;
                delta +=2;
        }

        return ((delta/2)-1);
}

int main()
{

        int num;


        cout<<endl<<"Enter the number to find square::::";
        cin>>num;

        cout<<endl<<"Square Root is::::"<<sqrt32(num)<<endl;
        return 0;
```

Prime Number Program using C++

```
#include<iostream>
using namespace std;


int main()
{
        int num;
        int count=2;
        cout<<endl<<"Enter the Number:::";
        cin>>num;

        for(;count<=num-1;count++)
        {
                if(num % count ==0)
                {
                        cout<<endl<<"Entered Number Is Not Prime Number"<<endl;
                        break;
                }
        }

        if(num==count)
        {
                cout<<endl<<"Entered Number Is Prime Number"<<endl;
        }

}
}
```

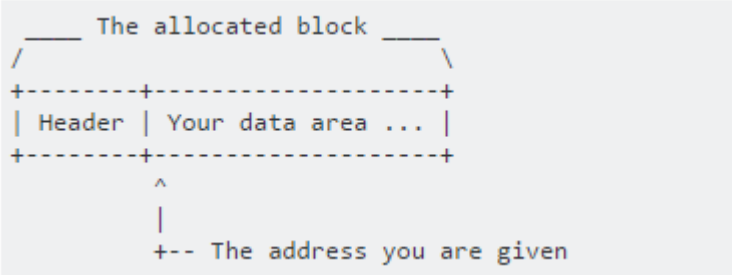
How does free() function knows how memory need to be deleted?

One typical way (in-line) is to actually allocate both a header and the memory you asked for, padded out to some minimum size. So for example, if you asked for 20 bytes, the system may allocate a 48-byte block:

1. 16-byte header containing size, special marker, checksum, pointers to next/previous block and so on.

2. 32 bytes data area (your 20 bytes padded out to a multiple of 16).

The address then given to you is the address of the data area. Then, when you free the block, freewill simply take the address you give it and, assuming you haven't stuffed up that address or the memory around it, check the accounting information immediately before it. Graphically, that would be along the lines of:



Will the below program work without error?

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

int main(void)
{
	int *ptr=(int*)malloc(sizeof(int)*10);
	ptr++;
	free(ptr);

	return EXIT_SUCCESS;
}
```

Output

```
root@jantu-Inspiron-3521:/home/jantu# ./a.out
*** glibc detected *** ./a.out: free(): invalid pointer: 0x0000000001b31014 ***
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x7da26)[0x7f336b2fca26]
./a.out[0x40056b]
/lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xed)[0x7f336b2a07ed]
./a.out[0x400489]
======= Memory map: ========
00400000-00401000 r-xp 00000000 08:03 4426005                            /home/jantu/a.out
00600000-00601000 r--p 00000000 08:03 4426005                            /home/jantu/a.out
00601000-00602000 rw-p 00001000 08:03 4426005                            /home/jantu/a.out
01b31000-01b52000 rw-p 00000000 00:00 0                                  [heap]
7f336b069000-7f336b07e000 r-xp 00000000 08:03 4375537                    /lib/x86_64-linux-gnu/libgcc_s.so.1
7f336b07e000-7f336b27d000 ---p 00015000 08:03 4375537                    /lib/x86_64-linux-gnu/libgcc_s.so.1
7f336b27d000-7f336b27e000 r--p 00014000 08:03 4375537                    /lib/x86_64-linux-gnu/libgcc_s.so.1
7f336b27e000-7f336b27f000 rw-p 00015000 08:03 4375537                    /lib/x86_64-linux-gnu/libgcc_s.so.1
7f336b27f000-7f336b433000 r-xp 00000000 08:03 4375320                    /lib/x86_64-linux-gnu/libc-2.15.so
7f336b433000-7f336b632000 ---p 001b4000 08:03 4375320                    /lib/x86_64-linux-gnu/libc-2.15.so
7f336b632000-7f336b636000 r--p 001b3000 08:03 4375320                    /lib/x86_64-linux-gnu/libc-2.15.so
7f336b636000-7f336b638000 rw-p 001b7000 08:03 4375320                    /lib/x86_64-linux-gnu/libc-2.15.so
7f336b638000-7f336b63d000 rw-p 00000000 00:00 0
7f336b63d000-7f336b65f000 r-xp 00000000 08:03 4375593                    /lib/x86_64-linux-gnu/ld-2.15.so
7f336b839000-7f336b83c000 rw-p 00000000 00:00 0
7f336b85c000-7f336b85f000 rw-p 00000000 00:00 0
7f336b85f000-7f336b860000 r--p 00022000 08:03 4375593                    /lib/x86_64-linux-gnu/ld-2.15.so
7f336b860000-7f336b862000 rw-p 00023000 08:03 4375593                    /lib/x86_64-linux-gnu/ld-2.15.so
7fffd5b14000-7fffd5b35000 rw-p 00000000 00:00 0                          [stack]
7fffd5b5a000-7fffd5b5c000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
Aborted (core dumped)
root@jantu-Inspiron-3521:/home/jantu#
```

How flip or toggle a bit of a number

The XOR operator (^) can be used to toggle a bit.

```
number ^= 1 << x;
```

That will toggle bit x

How to clear bit of a number

Use the bitwise AND operator (&) to clear a bit.

```
number &= ~(1 << x);
```

That will clear bit x. You must invert the bit string with the bitwise NOT operator (~), then AND it

How to set bit of a number

Use the bitwise OR operator (|) to set a bit.

```
number |= 1 << x;
```

That will set bit x

How reverse or swap bits of a number

```
unsigned char reverse(unsigned char b) {
  b = (b & 0xF0) >> 4 | (b & 0x0F) << 4;
  b = (b & 0xCC) >> 2 | (b & 0x33) << 2;
  b = (b & 0xAA) >> 1 | (b & 0x55) << 1;
  return b;
}
```

First the left four bits are swapped with the right four bits. Then all adjacent pairs are swapped and then

all adjacent single bits. This results in a reversed order.

```
unsigned int toReverse;
unsigned int reversed;
unsigned char inByte0 = (toReverse & 0xFF);
unsigned char inByte1 = (toReverse & 0xFF00) >> 8;
unsigned char inByte2 = (toReverse & 0xFF0000) >> 16;
unsigned char inByte3 = (toReverse & 0xFF000000) >> 24;
reversed = (reverseBits(inByte0) << 24) | (reverseBits(inByte1) << 16) |
(reverseBits(inByte2) << 8) | (reverseBits(inByte3);
```

Stack implementation using Link list

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

typedef struct node
{
        struct node *next;
        int data;
}NODE;

NODE *top=NULL;

void push_stack(int data)
```

```
{
        NODE *temp=(NODE*)malloc(sizeof(struct node)*1);

        if(top==NULL)
        {
                top=temp;
                top->data=data;
                top->next=NULL;
        }
        else
        {
                temp->next=top;
                temp->data=data;
                top=temp;
        }
} // Push of Stack

void pop_stack(void)
{
        NODE *temp;
        if(top==NULL)
        {
                printf("\n No Node Exit in the Node \n");
        }
        else if(top->next==NULL)
        {
                free(top);
                top=NULL;
        }
        else
        {
                temp=top;
                top=top->next;
                free(temp);
        }

}

void display_stack(void)
{
        NODE *trav;
        printf("\n Entered Stack Is:::::");

        if(top==NULL)
        {
                printf("\n Entered List Is Empty \n");
        }
        else
        {
                trav=top;
                while(trav !=NULL)
                {
                        printf("%d\t",trav->data);
                                trav=trav->next;
                }
```

```
        }
}// display of stack
int main(void)
{
        push_stack(400);
        push_stack(300);
        push_stack(200);
        push_stack(100);

        display_stack();

        pop_stack();

        display_stack();

        return 0;
}
```

Queue implementation using Link List

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

typedef struct node
{
        struct node *next;
        int data;
}NODE;

NODE *rear=NULL,*front=NULL;

void insert_into_queaue(int data)
{
        NODE *temp=(NODE*)malloc(sizeof(struct node)*1);
        NODE *trav;

        if(rear==NULL)
        {
                rear=front=temp;
                rear->data=data;
                rear->next=NULL;
        }
        else
        {
                trav=rear;
                while(trav->next !=NULL)
                {
                        trav=trav->next;
                }
                temp->next=NULL;
                temp->data=data;
```

```
                trav->next=temp;
                rear=temp;
        }
} // insert into queue

void delete_from_queue(void)
{
        NODE *temp;
        if(front==NULL)
        {
                printf("\n Queue Is Empty \n");
        }
        else if(front->next==NULL)
        {
                free(front);
                front=rear=NULL;
        }
        else
        {
                temp=front;
                front=front->next;
                free(temp);
        }
} // delete from queue

void display_of_queue(void)
{
        NODE *trav;
        printf("\n Entered queue Is:::::");

        if(front==NULL)
        {
                printf("\n Queue is Empty \n");
        }
        else
        {
                trav=front;

                while(trav !=NULL)
                {
                        printf("%d\t",trav->data);
                        trav=trav->next;
                }

        }
}// display of queue


int main(void)
{
        insert_into_queaue(100);
        insert_into_queaue(200);
        insert_into_queaue(300);
        insert_into_queaue(400);
        display_of_queue();
```

```
        delete_from_queue();
        display_of_queue();
        return 0;
}
```

Single malloc() call to allocate 2-D array

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

#define ROW 3
#define COL 3


int main(void)
{
        int i,j;
        int *data;
        int row_bytes=sizeof(int)*ROW;
        int col_bytes=sizeof(int)*COL;

        int **ptr=(int**)malloc(row_bytes+col_bytes*ROW);
        data=(int*)(ptr+ROW);

        for(i=0;i<ROW;i++)
        {
                ptr[i]=data+COL*i;
        }

        for(i=0;i<ROW;i++)
        {
                for(j=0;j<COL;j++)
                {
                        ptr[i][j]=i+j;
                }
        }

        printf("\n Entered Array Is::::\n");
        for(i=0;i<ROW;i++)
        {
                for(j=0;j<COL;j++)
                {
                        printf("%d\t",ptr[i][j]);
                }
                printf("\n");
        }

        return 0;
}
```

How to convert integer number to binary number

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

int main(void)
{

	unsigned int number=7;
	int i=(sizeof(int)*8) - 1;
	for(;i>=0;i--)
	{
		printf("%d",(number & 1<<i)?1:0);
	}

	printf("\n");

	return 0;
}
```

How to check whether given number is divisible by 8 using bitwise operator

```
#include <iostream>
using namespace std;

int main()
{
	unsigned int num;
	cout<<endl<<"Enter the number:::";
	cin>>num;

	if(((num>>3) <<3)==num)
	{
		cout<<endl<<"Number is divisible by 8"<<endl;
	}
	else
	{
		cout<<endl<<"Number is not divisible by 8"<<endl;
	}

return 0;
}
```

How to check whether a given integer number is divisible by 4 using bitwise operator

```
#include <iostream>
using namespace std;

int main()
{
	unsigned int num;
```

```
        cout<<endl<<"Enter the number:::";
        cin>>num;

        if(((num>>2) <<2)==num)
        {
                cout<<endl<<"Number is divisible by 4"<<endl;
        }
        else
        {
                cout<<endl<<"Number is not divisible by 4"<<endl;
        }

return 0;
}
```

Write one line C function to find whether a no is power of two

Given a positive integer, write a function to find if it is a power of two or not.

Examples:

```
Input : n = 4

Output : Yes

2^2 = 4


Input : n = 7

Output : No


Input : n = 32

Output : Yes

2^5 = 32
```

```
if((num != 1) && (num & (num - 1))) { /* make num pow of 2 */ }
```

How to check if an integer is a power of 3?

```
while (n % 3 == 0) {
   n /= 3;
}
```

```
return n == 1;
```

Find whether a given number is a power of 4 or not

Another solution is to keep dividing the number by 4, i.e, do n = n/4 iteratively. In any iteration, if n%4 becomes non-zero and n is not 1 then n is not a power of 4, otherwise n is a power of 4.

```
#include<stdio.h>
#define bool int

/* Function to check if x is power of 4*/
bool isPowerOfFour(int n)
{
  if(n == 0)
    return 0;
  while(n != 1)
  {
   if(n%4 != 0)
      return 0;
    n = n/4;
  }
  return 1;
}

/*Driver program to test above function*/
int main()
{
  int test_no = 64;
  if(isPowerOfFour(test_no))
    printf("%d is a power of 4", test_no);
  else
    printf("%d is not a power of 4", test_no);
  getchar();
}
```

How to check whether a linked list contains a cycle

Algorithm to detect cycle in a linked list
Let "head" be the head pointer of given linked list.

- Let, "slow" and "fast" be two node pointers pointing to the head node of linked list.
- In every iteration, the "slow" pointer moves ahead by one node(slow = slow->next;) whereas "fast" pointer moves two nodes at a time(fast = fast->next->next;).
- If linked list contains a loop, "slow" and "fast" pointers will eventually meet at the same node, thus indicating that the linked list contains a loop.

- If pointers do not meet then linked list doesn't have loop.

This algorithm is known as Floyd's Cycle-Finding Algorithm

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

/* A structure of linked list node */
struct node {
  int data;
  struct node *next;
} *head;

void initialize(){
    head = NULL;
}

/*
Given a Inserts a node in front of a singly linked list.
*/
void insert(int num) {
    /* Create a new Linked List node */
    struct node* newNode = (struct node*) malloc(sizeof(struct node));
    newNode->data  = num;
    /* Next pointer of new node will point to head node of linked list  */
    newNode->next = head;
    /* make new node as new head of linked list */
    head = newNode;
    printf("Inserted Element : %d\n", num);
}

void findloop(struct node *head) {
    struct node *slow, *fast;
    slow = fast = head;

    while(slow && fast && fast->next) {
        /* Slow pointer will move one node per iteration whereas
        fast node will move two nodes per iteration */
        slow = slow->next;
        fast  = fast->next->next;
        if (slow == fast) {
            printf("Linked List contains a loop\n");
            return;
        }
    }
    printf("No Loop in Linked List\n");
}
/*
 Prints a linked list from head node till tail node
*/
void printLinkedList(struct node *nodePtr) {
  while (nodePtr != NULL) {
     printf("%d", nodePtr->data);
     nodePtr = nodePtr->next;
     if(nodePtr != NULL)
```

```
        printf("-->");
  }
}

int main() {
   initialize();
   /* Creating a linked List*/
   insert(8);
   insert(3);
   insert(2);
   insert(7);
   insert(9);

   /* Create loop in linked list. Set next pointer of last node to second node from head */
   head->next->next->next->next->next = head->next;

   findloop(head);1111
   return 0;
}
```

Output

```
Inserted Element : 8
Inserted Element : 3
Inserted Element : 2
Inserted Element : 7
Inserted Element : 9
Linked List contains a loop
```

How to store integer hex value into string

```
int intval ;/*your value*/
char hexval[5];
sprintf(hexval,"%0x",intval);
```

Now use hexval[0] thru hexval[3]; if you want to use it as a null-terminated string then add

```
hexval[4]=0;
```

Your integer may contain more than four hex digits worth of data, hence the check first.

If you're not allowed to use library functions, divide it down into nybbles manually:

```
#define TO_HEX(i) (i <= 9 ? '0' + i : 'A' - 10 + i)

int x = 1234;

char res[5];

if (x <= 0xFFFF)
{
        res[0] = TO_HEX(((x & 0xF000) >> 12));
        res[1] = TO_HEX(((x & 0x0F00) >> 8));
        res[2] = TO_HEX(((x & 0x00F0) >> 4));
        res[3] = TO_HEX((x & 0x000F));
        res[4] = '\0';
}
```

How to store hex string into integer

```
uint32_t hex2int(char *hex) {
    uint32_t val = 0;
    while (*hex) {
        // get current character then increment
        uint8_t byte = *hex++;
        // transform hex character to the 4bit equivalent number, using the ascii table indexes
        if (byte >= '0' && byte <= '9') byte = byte - '0';
        else if (byte >= 'a' && byte <='f') byte = byte - 'a' + 10;
        else if (byte >= 'A' && byte <='F') byte = byte - 'A' + 10;
        // shift 4 to make space for new digit, and add the 4 bits of the new digit
        val = (val << 4) | (byte & 0xF);
    }
    return val;
}
```

```
int hex_to_int(unsigned char hex[], int count)
{
  int sum = 0;
  int i;
  int temp;

  for (i = 0; i < count; i++)
  {
    temp = (hex[i] & 0xf0U) >> 4;
    sum = sum*16 + temp;
    temp = (hex[i] & 0xofU);
    sum = sum*16 + temp;
  }
  return sum;
}
```

Write a program to generate 0, 1, 2, 5, 26… series.

```
#include<iostream>
using namespace std;


int main(void)
{
	int next=0;
	int num;
	cout<<endl<<"Number of Series Element:::";
	cin>>num;

	cout<<endl<<"Series IS::::";

	for(int i=0;i<num;i++)
	{
		if(i<=2)
		{
			next=i;
		}
		else
		{
			next=next*next+1;
		}
		cout<<next<<"\t";
	}

	return 0;
}
```