Stack

//imagine stack of books which is wrong but rather think of it like a tray

With a tray to get something in the center, you’ll have to remove the one on the top of the stack.

Stack is like an array

|  |
| --- |
| a[4] |
| a[3] |
| a[2] |
| a[1] |
| a[0] |

//push to add something to the stack and pop to remove from the stack

Counter {=0 stack is empty

The counter

{=5 stack is full

To clear the stack you have to set the counter to 0

If we drop something it will go to a[0] location

The counter now has to change because we have 1 item in the stack

|  |
| --- |
| a[4] |
| a[3] |
| a[2] 30 |
| a[1] 10 |
| a[0] 5 |

The counter is now 1 we add 10

Now the counter is 2 we add 30

Now the counter is 3

This is the process of adding /pushing into the stack

Class STACK

{

Private: int a[5]; //integer array of size 5

//we also need the counter

Int counter;

Public: //we need some members to do this stacking job for us

//we need one to clean the stack first

//so we set counter to 0

Void clearStack()

{counter=0;} //before we start stack we must call

//this to clean the stack first

//if we want to add something to the stackm we gotta

//make sure the stack isn’t full

//we want to write a function too if the stack is empty

Bool emptyStack() {//we look at the value of the counter if //it’s 0 , if not return false

if (counter==0) return true;

else return false)}

//we need another to see if stack is full

bool fullStack() {//if the counter is == to the size of the stack

if (counter==5) return true;

else return false;}

//if we want to push 7 for example, we want a function to put

//it on the stop of the stack

void pushStack(int x)

{

a[counter]=x;

counter++;

}

//next is to remove an item from the stack

//we’ll have to remove the top element of the stack, our counter is index 5 if it’s full, if we want to remove counter[4]

//we have to reduce the counter by 1

int popStack()

{

counter--; return a[counter]; //a[counter] is now currently //a[4] once it’s returned, we have access to the value in a[4]

//if we call this function another time, the counter is reduced //and we’re returned a[3]}

//the source of having access to the stack is the counter

} //this is the STACK CLASS

int main()

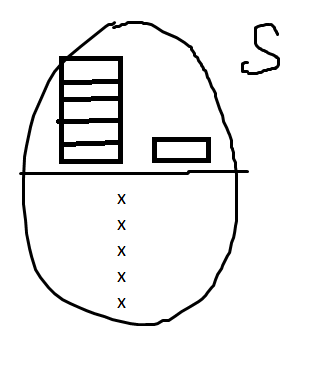
{

//we now want to create an object of STACK

STACK s; //we now have an object of STACK called s

//in the private section of this object we have array of int

//size 5



//before we use the stack we have to clear it

s.clearStack(); //now the counter is 0

//we want to drop something into the stack

|  |
| --- |
| a[4] |
| a[3] |
| a[2] |
| a[1] 20 |
| a[0] 10 |

s.pushStack(10); //before this counter was 0 and we //added 10 to it after this call counter is now 1

//the counter is now 1

s.pushStack(20); //counter is now 2

cout<<s.popStack(); //reduce counter by 1

//the counter was 2, now the counter is 1

//return a[1] so the first output is 20

cout<<s.popStack(); //reduce the counter by 1 again

//counter becomes 0 and return a[0]

//the output is now 10 which was at a[0]

//the main application of a stack is to reverse the order of…

}

//ANOTHER EXAMPLE

25base10 = //to change it to base 2 or binary, write the powers of 2

2^0 = 1

2^1 = 2

2^2 = 4

2^3 = 8

2^4 = 16

now 25 minus 16 we have 9, now we have 8 in the powers so we have 1

so in binary it’s

= 16 8 4 2 1

1 1 0 0 1

1 1 0 0 1

25 / 2 = 12 / 2 = 6 / 2 = 3 / 2 = 1 / 2 = 0

remainder 1 0 0 1 1

int main()

{

STACK s;

int n,r;

s.clearStack();

cout<<”Enter a number”<<endl;

cin>>n;

while(n!=0)

{

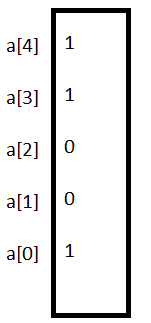
r=n%2; s.pushStack(r);

n=n/2; //this formula is the same as the remainder equation u

// wrote up there

}//end while

//pop the stack and print or display items



while(!s.emptyStack())//while not stack is empty then do this

{

int x = s.popStack();

cout<<x;

//the stack gets reduced from it being filled from

..previos while loop while it was being pushed

}

//this was a method or formula to return things or nmbers in binary

}

//FOR THE PROJECT PAY ATTENTION

25 base 10 if you convert it to base 2 will be = 1 1 0 0 1

Number Systems\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_digits\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_base

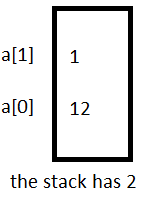
Decimal 0,1,2,3….,9 10

Binary 0,1 2

Octal 0,1,2. . .7 8

Hexadecimal 0,1,2. . .16 16

once 10 is reached, A – F



28 base 10 is what in X base 16

28/16 = 1 / 16 = 0

remainder: 12 1

while(!s.emptyStack())

{

int x=s.popStack();

if(x<=9) cout<<x;

else switch(x)

{

case 10: cout<<’A’; break;

case 11: cout<<’B’; break;

. . . . . . and so on

case 15: cout<<’F’; break;

}

}

//FOR PROJECT AGAIN!

//to generate random numbers/INTEGERS

x=rand(); x=0. . . .(max int)

x=rand()%10; //modulus 10. . . x=0, 1,. . . 9

x=rand()%100 x=0,1,2,3. . . 99

x=rand()%10+1; x=1,2. . . 10

zero to nine 0- 9

lets say we want to generate 10 random numbers <= 100

srand(time(0)); //going to use system clack to generate first random //number and will keep using to generate random numbers all the time

for(int i=1; i<=10; ++i)

{

x=rand()%100+1;

cout<<x<<’\t’; //we will still see the same random numbers, we want //the first number to change its value every time and that is srand to //seed random generator

}

//rand and srand are members of the <iostream> library

//time is a member of <ctime>