

Program for binomial heap

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

```
#include<malloc.h>
```

```
struct node {  
    int n;  
    int degree;  
    struct node* parent;  
    struct node* child;  
    struct node* sibling;  
};
```

```
struct node* MAKE_bin_HEAP();  
int bin_LINK(struct node*, struct node*);  
struct node* CREATE_NODE(int);  
struct node* bin_HEAP_UNION(struct node*, struct node*);  
struct node* bin_HEAP_INSERT(struct node*, struct node*);  
struct node* bin_HEAP_MERGE(struct node*, struct node*);  
struct node* bin_HEAP_EXTRACT_MIN(struct node*);  
int REVERT_LIST(struct node*);  
int DISPLAY(struct node*);  
struct node* FIND_NODE(struct node*, int);  
int bin_HEAP_DECREASE_KEY(struct node*, int, int);  
int bin_HEAP_DELETE(struct node*, int);
```

```
int count = 1;
```

```
struct node* MAKE_bin_HEAP() {  
    struct node* np;  
    np = NULL;  
    return np;  
}
```

```
}
```

```
struct node * H = NULL;
```

```
struct node *Hr = NULL;
```

```
int bin_LINK(struct node* y, struct node* z) {
```

```
    y->parent = z;
```

```
    y->sibling = z->child;
```

```
    z->child = y;
```

```
    z->degree = z->degree + 1;
```

```
}
```

```
struct node* CREATE_NODE(int k) {
```

```
    struct node* p;//new node;
```

```
    p = (struct node*) malloc(sizeof(struct node));
```

```
    p->n = k;
```

```
    return p;
```

```
}
```

```
struct node* bin_HEAP_UNION(struct node* H1, struct node* H2) {
```

```
    struct node* prev_x;
```

```
    struct node* next_x;
```

```
    struct node* x;
```

```
    struct node* H = MAKE_bin_HEAP();
```

```
    H = bin_HEAP_MERGE(H1, H2);
```

```
    if (H == NULL)
```

```
        return H;
```

```
    prev_x = NULL;
```

```
    x = H;
```

```
    next_x = x->sibling;
```

```
    while (next_x != NULL) {
```

```

if ((x->degree != next_x->degree) || ((next_x->sibling != NULL)
    && (next_x->sibling->degree == x->degree)) {
    prev_x = x;
    x = next_x;
} else {
    if (x->n <= next_x->n) {
        x->sibling = next_x->sibling;
        bin_LINK(next_x, x);
    } else {
        if (prev_x == NULL)
            H = next_x;
        else
            prev_x->sibling = next_x;
        bin_LINK(x, next_x);
        x = next_x;
    }
}
next_x = x->sibling;
}
return H;
}

```

```

struct node* bin_HEAP_INSERT(struct node* H, struct node* x) {
    struct node* H1 = MAKE_bin_HEAP();
    x->parent = NULL;
    x->child = NULL;
    x->sibling = NULL;
    x->degree = 0;
    H1 = x;
    H = bin_HEAP_UNION(H, H1);
    return H;
}

```

```
}
```

```
struct node* bin_HEAP_MERGE(struct node* H1, struct node* H2) {  
    struct node* H = MAKE_bin_HEAP();  
    struct node* y;  
    struct node* z;  
    struct node* a;  
    struct node* b;  
    y = H1;  
    z = H2;  
    if (y != NULL) {  
        if (z != NULL && y->degree <= z->degree)  
            H = y;  
        else if (z != NULL && y->degree > z->degree)  
            /* need some modifications here;the first and the else conditions can be merged together!!!!  
*/  
            H = z;  
        else  
            H = y;  
    } else  
        H = z;  
    while (y != NULL && z != NULL) {  
        if (y->degree < z->degree) {  
            y = y->sibling;  
        } else if (y->degree == z->degree) {  
            a = y->sibling;  
            y->sibling = z;  
            y = a;  
        } else {  
            b = z->sibling;  
            z->sibling = y;  
        }  
    }  
}
```

```

        z = b;
    }
}
return H;
}

```

```

int DISPLAY(struct node* H) {
    struct node* p;
    if (H == NULL) {
        printf("\nHEAP EMPTY");
        return 0;
    }
    printf("\nTHE ROOT NODES ARE:-\n");
    p = H;
    while (p != NULL) {
        printf("%d", p->n);
        if (p->sibling != NULL)
            printf("-->");
        p = p->sibling;
    }
    printf("\n");
}

```

```

struct node* bin_HEAP_EXTRACT_MIN(struct node* H1) {
    int min;
    struct node* t = NULL;
    struct node* x = H1;
    struct node *Hr;
    struct node* p;
    Hr = NULL;
    if (x == NULL) {

```

```

    printf("\nNOTHING TO EXTRACT");
    return x;
}
// int min=x->n;
p = x;
while (p->sibling != NULL) {
    if ((p->sibling)->n < min) {
        min = (p->sibling)->n;
        t = p;
        x = p->sibling;
    }
    p = p->sibling;
}
if (t == NULL && x->sibling == NULL)
    H1 = NULL;
else if (t == NULL)
    H1 = x->sibling;
else if (t->sibling == NULL)
    t = NULL;
else
    t->sibling = x->sibling;
if (x->child != NULL) {
    REVERT_LIST(x->child);
    (x->child)->sibling = NULL;
}
H = bin_HEAP_UNION(H1, Hr);
return x;
}

```

```

int REVERT_LIST(struct node* y) {
    if (y->sibling != NULL) {

```

```

    REVERT_LIST(y->sibling);
    (y->sibling)->sibling = y;
} else {
    Hr = y;
}
}

```

```

struct node* FIND_NODE(struct node* H, int k) {
    struct node* x = H;
    struct node* p = NULL;
    if (x->n == k) {
        p = x;
        return p;
    }
    if (x->child != NULL && p == NULL) {
        p = FIND_NODE(x->child, k);
    }

    if (x->sibling != NULL && p == NULL) {
        p = FIND_NODE(x->sibling, k);
    }
    return p;
}

```

```

int bin_HEAP_DECREASE_KEY(struct node* H, int i, int k) {
    int temp;
    struct node* p;
    struct node* y;
    struct node* z;
    p = FIND_NODE(H, i);
    if (p == NULL) {

```

```

    printf("\nINVALID CHOICE OF KEY TO BE REDUCED");
    return 0;
}
if (k > p->n) {
    printf("\nSORRY!THE NEW KEY IS GREATER THAN CURRENT ONE");
    return 0;
}
p->n = k;
y = p;
z = p->parent;
while (z != NULL && y->n < z->n) {
    temp = y->n;
    y->n = z->n;
    z->n = temp;
    y = z;
    z = z->parent;
}
printf("\nKEY REDUCED SUCCESSFULLY!");
}

```

```

int bin_HEAP_DELETE(struct node* H, int k) {
    struct node* np;
    if (H == NULL) {
        printf("\nHEAP EMPTY");
        return 0;
    }

    bin_HEAP_DECREASE_KEY(H, k, -1000);
    np = bin_HEAP_EXTRACT_MIN(H);
    if (np != NULL)
        printf("\nNODE DELETED SUCCESSFULLY");
}

```



```
}
```

```
int main() {  
    int i, n, m, l;  
    struct node* p;  
    struct node* np;  
    char ch;  
    printf("\nEnter the number of elements:");  
    scanf("%d", &n);  
    printf("\nEnter the elements:\n");  
    for (i = 1; i <= n; i++) {  
        scanf("%d", &m);  
        np = CREATE_NODE(m);  
        H = bin_HEAP_INSERT(H, np);  
    }  
    DISPLAY(H);  
    do {  
        printf("\nMenu:-\n");  
        printf(  
            "\n1) INSERT AN ELEMENT\n2) EXTRACT THE MINIMUM KEY NODE\n3) DECREASE A NODE  
KEY\n4) DELETE A NODE\n5) QUIT\n");  
        scanf("%d", &l);  
        switch (l) {  
        case 1:  
            do {  
                printf("\nEnter the element to be inserted:");  
                scanf("%d", &m);  
                p = CREATE_NODE(m);  
                H = bin_HEAP_INSERT(H, p);  
                printf("\nNow the heap is:\n");  
                DISPLAY(H);  
            }
```

```

    printf("\nINSERT MORE(y/Y)= \n");

    fflush(stdin);

    scanf("%c", &ch);
} while (ch == 'Y' || ch == 'y');

break;
case 2:

```

```

do {

    printf("\nEXTRACTING THE MINIMUM KEY NODE");

    p = bin_HEAP_EXTRACT_MIN(H);

    if (p != NULL)

        printf("\nTHE EXTRACTED NODE IS %d", p->n);

    printf("\nNOW THE HEAP IS:\n");

    DISPLAY(H);

    printf("\nEXTRACT MORE(y/Y)\n");

    fflush(stdin);

    scanf("%c", &ch);

} while (ch == 'Y' || ch == 'y');

break;

```

```

case 3:

do {

    printf("\nENTER THE KEY OF THE NODE TO BE DECREASED:");

    scanf("%d", &m);

    printf("\nENTER THE NEW KEY : ");

    scanf("%d", &l);

    bin_HEAP_DECREASE_KEY(H, m, l);

    printf("\nNOW THE HEAP IS:\n");

    DISPLAY(H);

    printf("\nDECREASE MORE(y/Y)\n");

    fflush(stdin);

    scanf("%c", &ch);

} while (ch == 'Y' || ch == 'y');

```

```
break;
```

case 4:

```
do {  
    printf("\nEnter the key to be deleted: ");  
    scanf("%d", &m);  
    bin_HEAP_DELETE(H, m);  
    printf("\nDelete more(y/Y)\n");  
    fflush(stdin);  
    scanf("%c", &ch);  
} while (ch == 'y' || ch == 'Y');  
break;
```

case 5:

```
printf("\nexit\n");  
break;
```

default:

```
printf("\nInvalid entry...try again....\n");  
}
```

```
} while (l != 5);
```

```
}
```

The screenshot shows a web-based IDE interface. On the left is a sidebar with navigation links: 'My Projects', 'Classroom' (with a 'new' badge), 'Learn Programming', 'Programming Questions', 'We are Hiring', 'Sign Up', and 'Login'. Below these are social media icons for Facebook and Twitter, and a '+ 39.4K' button. At the bottom of the sidebar are links for 'About', 'FAQ', 'Blog', 'Terms of Use', and 'Contact Us'. The main area is split into two panes. The top pane shows C code for a binary heap implementation, including functions for inserting, extracting the minimum, decreasing a key, and deleting a node. The bottom pane shows the program's execution output, which includes a menu of options (1) INSERT AN ELEMENT, (2) EXTRACT THE MINIMUM KEY NODE, (3) DECREASE A NODE KEY, (4) DELETE A NODE, (5) QUIT, and the user's choice of option 4, followed by the deletion of a node and the program's exit.

```
299         printf("\nEnter the new key : ");  
300         scanf("%d", &l);  
301         bin_HEAP_DECREASE_KEY(H, m, l);  
302         printf("\nNow the heap is : ");  
303     }  
304 }  
305  
306 // Menu  
307  
308 int main()  
309 {  
310     int l;  
311     struct heap H;  
312     H = initHeap(10);  
313     while (1)  
314     {  
315         printf("\nMenu:-\n");  
316         printf("1) INSERT AN ELEMENT\n");  
317         printf("2) EXTRACT THE MINIMUM KEY NODE\n");  
318         printf("3) DECREASE A NODE KEY\n");  
319         printf("4) DELETE A NODE\n");  
320         printf("5) QUIT\n");  
321         printf("Enter your choice : ");  
322         scanf("%d", &l);  
323         switch (l)  
324         {  
325             case 1:  
326                 insertElement(H);  
327                 break;  
328             case 2:  
329                 extractMin(H);  
330                 break;  
331             case 3:  
332                 decreaseKey(H);  
333                 break;  
334             case 4:  
335                 deleteNode(H);  
336                 break;  
337             case 5:  
338                 printf("\nexit\n");  
339                 exit(0);  
340             default:  
341                 printf("\nInvalid choice of key to be reduced\n");  
342                 printf("Node deleted successfully\n");  
343                 printf("Delete more(y/Y)\n");  
344                 break;  
345         }  
346     }  
347 }  
348  
349 // Program finished with exit code 0  
350 // Press ENTER to exit console.
```