

```

1  1.#include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 100
5
6  int stack[MAX];
7  int top = -1;
8
9
10 void push() {
11     int value;
12     if (top == MAX - 1) {
13         printf("Stack Overflow! Cannot push element.\n");
14     } else {
15         printf("Enter the value to be pushed: ");
16         scanf("%d", &value);
17         top++;
18         stack[top] = value;
19         printf("Element %d pushed onto the stack.\n", value);
20     }
21 }
22
23 void pop() {
24     if (top == -1) {
25         printf("Stack Underflow! No element to pop.\n");
26     } else {
27         printf("Element %d popped from the stack.\n", stack[top]);
28         top--;
29     }
30 }
31
32 void display() {
33     if (top == -1) {
34         printf("Stack is empty.\n");
35     } else {
36         printf("Stack elements are:\n");
37         for (int i = top; i >= 0; i--) {
38             printf("%d\n", stack[i]);
39         }
40     }
41 }
42
43 int main() {
44     int choice;
45
46     while (1) {
47         printf("\nStack Operations Menu:\n");
48         printf("1. PUSH\n");
49         printf("2. POP\n");
50         printf("3. Display\n");
51         printf("4. Exit\n");
52         printf("Enter your choice: ");
53         scanf("%d", &choice);
54
55         switch (choice) {
56             case 1:
57                 push();
58                 break;
59             case 2:
60                 pop();
61                 break;
62             case 3:
63                 display();
64                 break;
65             case 4:
66                 printf("Exiting program. Goodbye!\n");

```

```

67         exit(0);
68     default:
69         printf("Invalid choice! Please try again.\n");
70     }
71 }
72
73 return 0;
74 }
75
76 2. #include <stdio.h>
77 #include <stdlib.h>
78
79 #define MAX 100
80
81 int queue[MAX];
82 int front = -1, rear = -1;
83
84
85 void enqueue() {
86     int value;
87     if (rear == MAX - 1) {
88         printf("Queue Overflow! Cannot enqueue element.\n");
89     } else {
90         printf("Enter the value to be enqueued: ");
91         scanf("%d", &value);
92         if (front == -1) {
93             front = 0;
94         }
95         rear++;
96         queue[rear] = value;
97         printf("Element %d enqueued into the queue.\n", value);
98     }
99 }
100
101 void dequeue() {
102     if (front == -1 || front > rear) {
103         printf("Queue Underflow! No element to dequeue.\n");
104     } else {
105         printf("Element %d dequeued from the queue.\n", queue[front]);
106         front++;
107         if (front > rear) {
108             front = rear = -1; // Reset the queue if empty
109         }
110     }
111 }
112
113 void display() {
114     if (front == -1) {
115         printf("Queue is empty.\n");
116     } else {
117         printf("Queue elements are:\n");
118         for (int i = front; i <= rear; i++) {
119             printf("%d ", queue[i]);
120         }
121         printf("\n");
122     }
123 }
124
125 int main() {
126     int choice;
127
128     while (1) {
129         printf("\nQueue Operations Menu:\n");
130         printf("1. Enqueue\n");
131         printf("2. Dequeue\n");
132         printf("3. Display\n");

```

```

133     printf("4. Exit\n");
134     printf("Enter your choice: ");
135     scanf("%d", &choice);
136
137     switch (choice) {
138         case 1:
139             enqueue();
140             break;
141         case 2:
142             dequeue();
143             break;
144         case 3:
145             display();
146             break;
147         case 4:
148             printf("Exiting program. Goodbye!\n");
149             exit(0);
150         default:
151             printf("Invalid choice! Please try again.\n");
152     }
153 }
154
155 return 0;
156 }
157 #include <stdio.h>
158 #include <stdlib.h>
159
160 #define MAX 100
161
162 int queue[MAX];
163 int front = -1, rear = -1;
164
165
166 void enqueue() {
167     int value;
168     if (rear == MAX - 1) {
169         printf("Queue Overflow! Cannot enqueue element.\n");
170     } else {
171         printf("Enter the value to be enqueued: ");
172         scanf("%d", &value);
173         if (front == -1) {
174             front = 0;
175         }
176         rear++;
177         queue[rear] = value;
178         printf("Element %d enqueued into the queue.\n", value);
179     }
180 }
181
182 void dequeue() {
183     if (front == -1 || front > rear) {
184         printf("Queue Underflow! No element to dequeue.\n");
185     } else {
186         printf("Element %d dequeued from the queue.\n", queue[front]);
187         front++;
188         if (front > rear) {
189             front = rear = -1; // Reset the queue if empty
190         }
191     }
192 }
193
194 void display() {
195     if (front == -1) {
196         printf("Queue is empty.\n");
197     } else {
198         printf("Queue elements are:\n");

```

```

199     for (int i = front; i <= rear; i++) {
200         printf("%d ", queue[i]);
201     }
202     printf("\n");
203 }
204 }
205
206 int main() {
207     int choice;
208
209     while (1) {
210         printf("\nQueue Operations Menu:\n");
211         printf("1. Enqueue\n");
212         printf("2. Dequeue\n");
213         printf("3. Display\n");
214         printf("4. Exit\n");
215         printf("Enter your choice: ");
216         scanf("%d", &choice);
217
218         switch (choice) {
219             case 1:
220                 enqueue();
221                 break;
222             case 2:
223                 dequeue();
224                 break;
225             case 3:
226                 display();
227                 break;
228             case 4:
229                 printf("Exiting program. Goodbye!\n");
230                 exit(0);
231             default:
232                 printf("Invalid choice! Please try again.\n");
233         }
234     }
235
236     return 0;
237 }
238 #include <stdio.h>
239 #include <stdlib.h>
240
241 #define MAX 100
242
243 int queue[MAX];
244 int front = -1, rear = -1;
245
246
247 void enqueue() {
248     int value;
249     if (rear == MAX - 1) {
250         printf("Queue Overflow! Cannot enqueue element.\n");
251     } else {
252         printf("Enter the value to be enqueued: ");
253         scanf("%d", &value);
254         if (front == -1) {
255             front = 0;
256         }
257         rear++;
258         queue[rear] = value;
259         printf("Element %d enqueued into the queue.\n", value);
260     }
261 }
262
263 void dequeue() {
264     if (front == -1 || front > rear) {

```

```

265     printf("Queue Underflow! No element to dequeue.\n");
266 } else {
267     printf("Element %d dequeued from the queue.\n", queue[front]);
268     front++;
269     if (front > rear) {
270         front = rear = -1; // Reset the queue if empty
271     }
272 }
273 }
274
275 void display() {
276     if (front == -1) {
277         printf("Queue is empty.\n");
278     } else {
279         printf("Queue elements are:\n");
280         for (int i = front; i <= rear; i++) {
281             printf("%d ", queue[i]);
282         }
283         printf("\n");
284     }
285 }
286
287 int main() {
288     int choice;
289
290     while (1) {
291         printf("\nQueue Operations Menu:\n");
292         printf("1. Enqueue\n");
293         printf("2. Dequeue\n");
294         printf("3. Display\n");
295         printf("4. Exit\n");
296         printf("Enter your choice: ");
297         scanf("%d", &choice);
298
299         switch (choice) {
300             case 1:
301                 enqueue();
302                 break;
303             case 2:
304                 dequeue();
305                 break;
306             case 3:
307                 display();
308                 break;
309             case 4:
310                 printf("Exiting program. Goodbye!\n");
311                 exit(0);
312             default:
313                 printf("Invalid choice! Please try again.\n");
314         }
315     }
316
317     return 0;
318 }
319
320 #include <stdio.h>
321 #include <stdlib.h>
322
323 #define MAX 100
324
325 int queue[MAX];
326 int front = -1, rear = -1;
327
328
329 void enqueue() {
330     int value;

```

```

331     if (rear == MAX - 1) {
332         printf("Queue Overflow! Cannot enqueue element.\n");
333     } else {
334         printf("Enter the value to be enqueued: ");
335         scanf("%d", &value);
336         if (front == -1) {
337             front = 0;
338         }
339         rear++;
340         queue[rear] = value;
341         printf("Element %d enqueued into the queue.\n", value);
342     }
343 }
344
345 void dequeue() {
346     if (front == -1 || front > rear) {
347         printf("Queue Underflow! No element to dequeue.\n");
348     } else {
349         printf("Element %d dequeued from the queue.\n", queue[front]);
350         front++;
351         if (front > rear) {
352             front = rear = -1; // Reset the queue if empty
353         }
354     }
355 }
356
357 void display() {
358     if (front == -1) {
359         printf("Queue is empty.\n");
360     } else {
361         printf("Queue elements are:\n");
362         for (int i = front; i <= rear; i++) {
363             printf("%d ", queue[i]);
364         }
365         printf("\n");
366     }
367 }
368
369 int main() {
370     int choice;
371
372     while (1) {
373         printf("\nQueue Operations Menu:\n");
374         printf("1. Enqueue\n");
375         printf("2. Dequeue\n");
376         printf("3. Display\n");
377         printf("4. Exit\n");
378         printf("Enter your choice: ");
379         scanf("%d", &choice);
380
381         switch (choice) {
382             case 1:
383                 enqueue();
384                 break;
385             case 2:
386                 dequeue();
387                 break;
388             case 3:
389                 display();
390                 break;
391             case 4:
392                 printf("Exiting program. Goodbye!\n");
393                 exit(0);
394             default:
395                 printf("Invalid choice! Please try again.\n");
396         }

```

```

397     }
398
399     return 0;
400 }
401
402
403
404 2. #include <stdio.h>
405 #include <stdlib.h>
406
407 #define MAX 100
408
409 int queue[MAX];
410 int front = -1, rear = -1;
411
412
413 void enqueue() {
414     int value;
415     if (rear == MAX - 1) {
416         printf("Queue Overflow! Cannot enqueue element.\n");
417     } else {
418         printf("Enter the value to be enqueued: ");
419         scanf("%d", &value);
420         if (front == -1) {
421             front = 0;
422         }
423         rear++;
424         queue[rear] = value;
425         printf("Element %d enqueued into the queue.\n", value);
426     }
427 }
428
429 void dequeue() {
430     if (front == -1 || front > rear) {
431         printf("Queue Underflow! No element to dequeue.\n");
432     } else {
433         printf("Element %d dequeued from the queue.\n", queue[front]);
434         front++;
435         if (front > rear) {
436             front = rear = -1; // Reset the queue if empty
437         }
438     }
439 }
440
441 void display() {
442     if (front == -1) {
443         printf("Queue is empty.\n");
444     } else {
445         printf("Queue elements are:\n");
446         for (int i = front; i <= rear; i++) {
447             printf("%d ", queue[i]);
448         }
449         printf("\n");
450     }
451 }
452
453 int main() {
454     int choice;
455
456     while (1) {
457         printf("\nQueue Operations Menu:\n");
458         printf("1. Enqueue\n");
459         printf("2. Dequeue\n");
460         printf("3. Display\n");
461         printf("4. Exit\n");
462         printf("Enter your choice: ");

```

```

463     scanf("%d", &choice);
464
465     switch (choice) {
466         case 1:
467             enqueue();
468             break;
469         case 2:
470             dequeue();
471             break;
472         case 3:
473             display();
474             break;
475         case 4:
476             printf("Exiting program. Goodbye!\n");
477             exit(0);
478         default:
479             printf("Invalid choice! Please try again.\n");
480     }
481 }
482
483 return 0;
484 }
485
486
487 3. #include <stdio.h>
488 #include <stdlib.h>
489
490 #define MAX 100
491
492 int queue[MAX];
493 int front = -1, rear = -1;
494
495
496 void enqueue() {
497     int value;
498     if (rear == MAX - 1) {
499         printf("Queue Overflow! Cannot enqueue element.\n");
500     } else {
501         printf("Enter the value to be enqueued: ");
502         scanf("%d", &value);
503         if (front == -1) {
504             front = 0;
505         }
506         rear++;
507         queue[rear] = value;
508         printf("Element %d enqueued into the queue.\n", value);
509     }
510 }
511
512 void dequeue() {
513     if (front == -1 || front > rear) {
514         printf("Queue Underflow! No element to dequeue.\n");
515     } else {
516         printf("Element %d dequeued from the queue.\n", queue[front]);
517         front++;
518         if (front > rear) {
519             front = rear = -1; // Reset the queue if empty
520         }
521     }
522 }
523
524 void display() {
525     if (front == -1) {
526         printf("Queue is empty.\n");
527     } else {
528         printf("Queue elements are:\n");

```



```

529         for (int i = front; i <= rear; i++) {
530             printf("%d ", queue[i]);
531         }
532         printf("\n");
533     }
534 }
535
536 int main() {
537     int choice;
538
539     while (1) {
540         printf("\nQueue Operations Menu:\n");
541         printf("1. Enqueue\n");
542         printf("2. Dequeue\n");
543         printf("3. Display\n");
544         printf("4. Exit\n");
545         printf("Enter your choice: ");
546         scanf("%d", &choice);
547
548         switch (choice) {
549             case 1:
550                 enqueue();
551                 break;
552             case 2:
553                 dequeue();
554                 break;
555             case 3:
556                 display();
557                 break;
558             case 4:
559                 printf("Exiting program. Goodbye!\n");
560                 exit(0);
561             default:
562                 printf("Invalid choice! Please try again.\n");
563         }
564     }
565
566     return 0;
567 }
568
569 #include <stdio.h>
570 #include <stdlib.h>
571
572 struct Node {
573     int data;
574     struct Node* next;
575 };
576
577 struct Node* head = NULL;
578
579 struct Node* createNode(int value) {
580     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
581     newNode->data = value;
582     newNode->next = NULL;
583     return newNode;
584 }
585
586
587 void insertAtBeginning() {
588     int value;
589     printf("Enter the value to insert at the beginning: ");
590     scanf("%d", &value);
591     struct Node* newNode = createNode(value);
592     newNode->next = head;
593     head = newNode;
594     printf("Node inserted at the beginning.\n");

```

```

595 }
596
597 void insertAtEnd() {
598     int value;
599     printf("Enter the value to insert at the end: ");
600     scanf("%d", &value);
601     struct Node* newNode = createNode(value);
602     if (head == NULL) {
603         head = newNode;
604     } else {
605         struct Node* temp = head;
606         while (temp->next != NULL) {
607             temp = temp->next;
608         }
609         temp->next = newNode;
610     }
611     printf("Node inserted at the end.\n");
612 }
613
614 void insertAtPosition() {
615     int value, pos;
616     printf("Enter the value to insert: ");
617     scanf("%d", &value);
618     printf("Enter the position to insert: ");
619     scanf("%d", &pos);
620
621     struct Node* newNode = createNode(value);
622     if (pos == 1) {
623         newNode->next = head;
624         head = newNode;
625     } else {
626         struct Node* temp = head;
627         for (int i = 1; i < pos - 1 && temp != NULL; i++) {
628             temp = temp->next;
629         }
630         if (temp == NULL) {
631             printf("Invalid position!\n");
632             free(newNode);
633             return;
634         }
635         newNode->next = temp->next;
636         temp->next = newNode;
637     }
638     printf("Node inserted at position %d.\n", pos);
639 }
640
641
642 void deleteFromBeginning() {
643     if (head == NULL) {
644         printf("List is empty. Nothing to delete.\n");
645     } else {
646         struct Node* temp = head;
647         head = head->next;
648         printf("Node with value %d deleted from the beginning.\n", temp->data);
649         free(temp);
650     }
651 }
652
653 void deleteFromEnd() {
654     if (head == NULL) {
655         printf("List is empty. Nothing to delete.\n");
656     } else if (head->next == NULL) {
657         printf("Node with value %d deleted from the end.\n", head->data);
658         free(head);
659         head = NULL;
660     } else {

```

```

661     struct Node* temp = head;
662     while (temp->next->next != NULL) {
663         temp = temp->next;
664     }
665     printf("Node with value %d deleted from the end.\n", temp->next->data);
666     free(temp->next);
667     temp->next = NULL;
668 }
669 }
670
671 void deleteFromPosition() {
672     int pos;
673     printf("Enter the position to delete: ");
674     scanf("%d", &pos);
675
676     if (head == NULL) {
677         printf("List is empty. Nothing to delete.\n");
678     } else if (pos == 1) {
679         struct Node* temp = head;
680         head = head->next;
681         printf("Node with value %d deleted from position 1.\n", temp->data);
682         free(temp);
683     } else {
684         struct Node* temp = head;
685         for (int i = 1; i < pos - 1 && temp != NULL; i++) {
686             temp = temp->next;
687         }
688         if (temp == NULL || temp->next == NULL) {
689             printf("Invalid position!\n");
690             return;
691         }
692         struct Node* toDelete = temp->next;
693         temp->next = toDelete->next;
694         printf("Node with value %d deleted from position %d.\n", toDelete->data, pos);
695         free(toDelete);
696     }
697 }
698
699 void search() {
700     int value, pos = 1;
701     printf("Enter the value to search: ");
702     scanf("%d", &value);
703
704     struct Node* temp = head;
705     while (temp != NULL) {
706         if (temp->data == value) {
707             printf("Element %d found at position %d.\n", value, pos);
708             return;
709         }
710         temp = temp->next;
711         pos++;
712     }
713     printf("Element %d not found in the list.\n", value);
714 }
715
716 void display() {
717     if (head == NULL) {
718         printf("List is empty.\n");
719     } else {
720         printf("List elements are: ");
721         struct Node* temp = head;
722         while (temp != NULL) {
723             printf("%d -> ", temp->data);
724             temp = temp->next;
725         }
726         printf("NULL\n");

```

```

727     }
728 }
729
730 int main() {
731     int choice;
732     while (1) {
733         printf("\nSingly Linked List Operations Menu:\n");
734         printf("1. Insert at Beginning\n");
735         printf("2. Insert at End\n");
736         printf("3. Insert at Position\n");
737         printf("4. Delete from Beginning\n");
738         printf("5. Delete from End\n");
739         printf("6. Delete from Position\n");
740         printf("7. Search\n");
741         printf("8. Display\n");
742         printf("9. Exit\n");
743         printf("Enter your choice: ");
744         scanf("%d", &choice);
745
746         switch (choice) {
747             case 1:
748                 insertAtBeginning();
749                 break;
750             case 2:
751                 insertAtEnd();
752                 break;
753             case 3:
754                 insertAtPosition();
755                 break;
756             case 4:
757                 deleteFromBeginning();
758                 break;
759             case 5:
760                 deleteFromEnd();
761                 break;
762             case 6:
763                 deleteFromPosition();
764                 break;
765             case 7:
766                 search();
767                 break;
768             case 8:
769                 display();
770                 break;
771             case 9:
772                 printf("Exiting program. Goodbye!\n");
773                 exit(0);
774             default:
775                 printf("Invalid choice! Please try again.\n");
776         }
777     }
778
779     return 0;
780 }
781
782 #include <stdio.h>
783 #include <stdlib.h>
784
785 struct Node {
786     int data;
787     struct Node* prev;
788     struct Node* next;
789 };
790
791 struct Node* head = NULL;
792

```

```

793
794 struct Node* createNode(int value) {
795     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
796     newNode->data = value;
797     newNode->prev = NULL;
798     newNode->next = NULL;
799     return newNode;
800 }
801
802 void insertAtBeginning() {
803     int value;
804     printf("Enter the value to insert at the beginning: ");
805     scanf("%d", &value);
806     struct Node* newNode = createNode(value);
807
808     if (head == NULL) {
809         head = newNode;
810     } else {
811         newNode->next = head;
812         head->prev = newNode;
813         head = newNode;
814     }
815     printf("Node inserted at the beginning.\n");
816 }
817
818 void insertAtEnd() {
819     int value;
820     printf("Enter the value to insert at the end: ");
821     scanf("%d", &value);
822     struct Node* newNode = createNode(value);
823
824     if (head == NULL) {
825         head = newNode;
826     } else {
827         struct Node* temp = head;
828         while (temp->next != NULL) {
829             temp = temp->next;
830         }
831         temp->next = newNode;
832         newNode->prev = temp;
833     }
834     printf("Node inserted at the end.\n");
835 }
836
837 void insertAtPosition() {
838     int value, pos;
839     printf("Enter the value to insert: ");
840     scanf("%d", &value);
841     printf("Enter the position to insert: ");
842     scanf("%d", &pos);
843
844     struct Node* newNode = createNode(value);
845
846     if (pos == 1) {
847         newNode->next = head;
848         if (head != NULL) {
849             head->prev = newNode;
850         }
851         head = newNode;
852     } else {
853         struct Node* temp = head;
854         for (int i = 1; i < pos - 1 && temp != NULL; i++) {
855             temp = temp->next;
856         }
857         if (temp == NULL) {
858             printf("Invalid position!\n");

```

```

859         free(newNode);
860         return;
861     }
862     newNode->next = temp->next;
863     newNode->prev = temp;
864     if (temp->next != NULL) {
865         temp->next->prev = newNode;
866     }
867     temp->next = newNode;
868 }
869 printf("Node inserted at position %d.\n", pos);
870 }
871
872 void deleteFromBeginning() {
873     if (head == NULL) {
874         printf("List is empty. Nothing to delete.\n");
875     } else {
876         struct Node* temp = head;
877         head = head->next;
878         if (head != NULL) {
879             head->prev = NULL;
880         }
881         printf("Node with value %d deleted from the beginning.\n", temp->data);
882         free(temp);
883     }
884 }
885
886 void deleteFromEnd() {
887     if (head == NULL) {
888         printf("List is empty. Nothing to delete.\n");
889     } else if (head->next == NULL) {
890         printf("Node with value %d deleted from the end.\n", head->data);
891         free(head);
892         head = NULL;
893     } else {
894         struct Node* temp = head;
895         while (temp->next != NULL) {
896             temp = temp->next;
897         }
898         printf("Node with value %d deleted from the end.\n", temp->data);
899         temp->prev->next = NULL;
900         free(temp);
901     }
902 }
903
904 void deleteFromPosition() {
905     int pos;
906     printf("Enter the position to delete: ");
907     scanf("%d", &pos);
908
909     if (head == NULL) {
910         printf("List is empty. Nothing to delete.\n");
911     } else if (pos == 1) {
912         struct Node* temp = head;
913         head = head->next;
914         if (head != NULL) {
915             head->prev = NULL;
916         }
917         printf("Node with value %d deleted from position 1.\n", temp->data);
918         free(temp);
919     } else {
920         struct Node* temp = head;
921         for (int i = 1; i < pos && temp != NULL; i++) {
922             temp = temp->next;
923         }
924         if (temp == NULL) {

```

```

925         printf("Invalid position!\n");
926         return;
927     }
928     printf("Node with value %d deleted from position %d.\n", temp->data, pos);
929     if (temp->prev != NULL) {
930         temp->prev->next = temp->next;
931     }
932     if (temp->next != NULL) {
933         temp->next->prev = temp->prev;
934     }
935     free(temp);
936 }
937 }
938
939 void search() {
940     int value, pos = 1;
941     printf("Enter the value to search: ");
942     scanf("%d", &value);
943
944     struct Node* temp = head;
945     while (temp != NULL) {
946         if (temp->data == value) {
947             printf("Element %d found at position %d.\n", value, pos);
948             return;
949         }
950         temp = temp->next;
951         pos++;
952     }
953     printf("Element %d not found in the list.\n", value);
954 }
955
956
957 void display() {
958     if (head == NULL) {
959         printf("List is empty.\n");
960     } else {
961         printf("List elements are: ");
962         struct Node* temp = head;
963         while (temp != NULL) {
964             printf("%d <-> ", temp->data);
965             temp = temp->next;
966         }
967         printf("NULL\n");
968     }
969 }
970
971 int main() {
972     int choice;
973
974     while (1) {
975         printf("\nDoubly Linked List Operations Menu:\n");
976         printf("1. Insert at Beginning\n");
977         printf("2. Insert at End\n");
978         printf("3. Insert at Position\n");
979         printf("4. Delete from Beginning\n");
980         printf("5. Delete from End\n");
981         printf("6. Delete from Position\n");
982         printf("7. Search\n");
983         printf("8. Display\n");
984         printf("9. Exit\n");
985         printf("Enter your choice: ");
986         scanf("%d", &choice);
987
988         switch (choice) {
989             case 1:
990                 insertAtBeginning();

```

```
991         break;
992     case 2:
993         insertAtEnd();
994         break;
995     case 3:
996         insertAtPosition();
997         break;
998     case 4:
999         deleteFromBeginning();
1000        break;
1001     case 5:
1002         deleteFromEnd();
1003         break;
1004     case 6:
1005         deleteFromPosition();
1006         break;
1007     case 7:
1008         search();
1009         break;
1010     case 8:
1011         display();
1012         break;
1013     case 9:
1014         printf("Exiting program. Goodbye!\n");
1015         exit(0);
1016     default:
1017         printf("Invalid choice! Please try again.\n");
1018     }
1019 }
1020
1021 return 0;
1022 }
```


1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122

1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188

1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254

1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320

1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386

1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452

1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518

1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562