

内容

| | | |
|-----|--------------------------------------|----|
| 1. | Linked Stack | 2 |
| 1.4 | Bracket Matching..... | 4 |
| 2.1 | Linked Queues by Linked Stacks | 5 |
| 2.2 | Linked Stacks by Linked Queues | 7 |
| 3. | Linked List..... | 9 |
| 4. | Tree | 12 |

1. Linked Stack

The image shows a C++ IDE with two windows. The left window displays the implementation of a Linked Stack, and the right window shows the output of the program.

Left Window: HMMLinkedStack.h

```
1 // HMMLinkedStack.h
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 // Enumeration for flags
6 typedef enum { FROM_TOP, FROM_BOTTOM } Direction;
7
8 typedef enum { RECOVER, NO_RECOVER } RecoveryOption;
9
10 // Define Node
11 typedef struct _Node {
12     int data;
13     NodePtr link;
14 } Node;
15
16 // LinkedStack functions
17 int empty(NodePtr *top) { return *top == NULL; }
18
19 void push(NodePtr *top, int data) {
20     Node new_node = (Node)malloc(sizeof(Node));
21     if (new_node == NULL) {
22         printf("Memory allocation error!\n");
23         exit(EXIT_FAILURE);
24     }
25     new_node->data = data;
26     new_node->link = *top;
27     *top = new_node;
28 }
29
30 NodePtr pop(NodePtr *top) {
31     if (empty(*top)) {
32         printf("Stack is empty!\n");
33         exit(EXIT_FAILURE);
34     }
35     NodePtr popped = *top;
36     *top = popped->link;
37     return popped;
38 }
39
40 void create_stack() { return NULL; }
41
42 void print(NodePtr top, int elements_in_one_line) {
43     NodePtr current = top;
44     int elements_in_current_line = 0;
45     while (current != NULL) {
46         printf("%d ", current->data);
47         current = current->link;
48         if (++elements_in_current_line == elements_in_one_line) {
49             printf("\n");
50             elements_in_current_line = 0;
51         }
52     }
53     printf(elements_in_current_line > 0 ? "\n" : "");
54 }
55
56 // Question
57
```

Right Window: Test Results

```
1. Use rand() to get 30 random numbers, output the numbers (one by one, one space in between, and 8 n
unders in one line) and push the numbers into S one by one
35 69 62 53 35 16 38 88
94 41 97 65 16 31 94 18
88 43 48 83 14 56 47 18
24 57 94 61 87 97

2. Assign and output integer x the 11th element from the top of S, leaving S unchanged.
x = 97
35 69 62 53 35 16 38 88
94 41 97 65 16 31 94 18
88 43 48 83 14 56 47 18
24 57 94 61 87 97

3. Put integer x in (2) under the bottom of S (leaving the rest of S unchanged) and output the n
s (one by one, one space in between, and 8 numbers in one line) on S from the top to the bottom
35 69 62 53 35 16 38 88
94 41 97 65 16 31 94 18
88 43 48 83 14 56 47 18
24 57 94 61 87 97

Process exited after 0.02335 seconds with return value 0
请按任意键继续 . . .
```

```
HHWLinkedList.c
33 |     elements_in_current_line = 0;
34 | }
35 |
36 | printf(elements_in_current_line ? "un" : "");
37 |
38 | // 41448222 Joinline
39 | // Question
40 | void random_number_generator(NodePtr *top, int range, int offset, int len) {
41 |     srand(time(NULL));
42 |     while (!empty(top)) pop(top);
43 |     int i;
44 |     for (i = 0; i < len; i++) push(top, rand() % range + offset);
45 | }
46 |
47 | // Helper function to reverse the stack
48 | void reverse_stack(NodePtr *top) {
49 |     NodePtr s1 = create_stack();
50 |     NodePtr s2 = create_stack();
51 |
52 |     // Move elements from the original stack to s1
53 |     while (!empty(top)) {
54 |         NodePtr popped = pop(top);
55 |         push(s1, popped->data);
56 |         free(popped);
57 |     }
58 |
59 |     // Move elements from s1 to s2
60 |     while (!empty(s1)) {
61 |         NodePtr popped = pop(s1);
62 |         push(s2, popped->data);
63 |         free(popped);
64 |     }
65 |
66 |     // Move elements from s2 back to the original stack
67 |     while (!empty(s2)) {
68 |         NodePtr popped = pop(s2);
69 |         push(top, popped->data);
70 |         free(popped);
71 |     }
72 | }
73 |
74 | // Helper function to recover elements to the original stack
75 | void recover_elements(NodePtr *top, NodePtr *temp_s) {
76 |     while (!empty(temp_s)) {
77 |         NodePtr popped = pop(temp_s);
78 |         push(top, popped->data);
79 |         free(popped);
80 |     }
81 | }
82 |
83 | // 41448222 Joinline
84 | // Helper function to get an element from the stack
85 | int get_element(NodePtr *top, Direction direction, int position, RecoveryOption recovery) {
86 |     NodePtr temp_s = create_stack();
87 |     int popped_value, i;
88 |
89 |     if (direction == FROM_TOP) {
90 |         for (i = 0; i < position; i++) {
91 |             NodePtr popped = pop(top);
92 |             push(temp_s, popped->data);
93 |             popped_value = popped->data;
94 |             free(popped);
95 |         }
96 |     }
97 |
98 |     if (recovery == RECOVER) recover_elements(top, temp_s);
99 |     else
100 |         reverse_stack(top); // reverse s to pop the bottom n-th element
101 |
102 |     for (i = 0; i < position; i++) {
103 |         NodePtr popped = pop(top);
104 |         push(temp_s, popped->data);
105 |         popped_value = popped->data;
106 |         free(popped);
107 |     }
108 |
109 |     if (recovery == RECOVER) recover_elements(top, temp_s);
110 |     reverse_stack(top); // reverse s to recover elements to the original direction
111 |
112 |     return popped_value;
113 | }
114 |
115 | // 41448222 Joinline
116 | int main() {
117 |     NodePtr s = create_stack();
118 |     int n;
119 |
120 |     // 1. Use rand() to get 10 random numbers, output the numbers (one by one, one space in between, and 8 numbers in one line) and push the numbers into S one by one
121 |     printf("1. Use rand() to get 10 random numbers, output the numbers (one by one, one space in between, and 8 numbers in one line) and push the numbers into S one by one\n");
122 |     random_number_generator(&s, 100, 1, 10);
123 |     printf("1. // 10 numbers per line\n");
124 |     printf("un\n");
125 |
126 |     // 2. Assign and output integer x the 11th element from the top of S, leaving S unchanged.
127 |     printf("2. Assign and output integer x the 11th element from the top of S, leaving S unchanged.\n");
128 |     int x = get_element(&s, FROM_TOP, 11, RECOVER);
129 |     printf("x = %d\n", x);
130 |     printf("un\n");
131 |
132 |     // 3. Put integer n in (2) under the bottom of S (leaving the rest of S unchanged) and output the numbers (one by one, one space in between, and 8 numbers in one line) on S from the top to the bottom.
133 |     reverse_stack(&s);
134 | }
```

```
HHWLinkedList.c
91 |     push(top, popped->data);
92 |     free(popped);
93 | }
94 |
95 | // Helper function to recover elements to the original stack
96 | void recover_elements(NodePtr *top, NodePtr *temp_s) {
97 |     while (!empty(temp_s)) {
98 |         NodePtr popped = pop(temp_s);
99 |         push(top, popped->data);
100 |         free(popped);
101 |     }
102 | }
103 |
104 | // 41448222 Joinline
105 | // Helper function to get an element from the stack
106 | int get_element(NodePtr *top, Direction direction, int position, RecoveryOption recovery) {
107 |     NodePtr temp_s = create_stack();
108 |     int popped_value, i;
109 |
110 |     if (direction == FROM_TOP) {
111 |         for (i = 0; i < position; i++) {
112 |             NodePtr popped = pop(top);
113 |             push(temp_s, popped->data);
114 |             popped_value = popped->data;
115 |             free(popped);
116 |         }
117 |     }
118 |
119 |     if (recovery == RECOVER) recover_elements(top, temp_s);
120 |     else
121 |         reverse_stack(top); // reverse s to pop the bottom n-th element
122 |
123 |     for (i = 0; i < position; i++) {
124 |         NodePtr popped = pop(top);
125 |         push(temp_s, popped->data);
126 |         popped_value = popped->data;
127 |         free(popped);
128 |     }
129 |
130 |     if (recovery == RECOVER) recover_elements(top, temp_s);
131 |     reverse_stack(top); // reverse s to recover elements to the original direction
132 |
133 |     return popped_value;
134 | }
135 |
136 | // 41448222 Joinline
137 | int main() {
138 |     NodePtr s = create_stack();
139 |     int n;
140 |
141 |     // 1. Use rand() to get 10 random numbers, output the numbers (one by one, one space in between, and 8 numbers in one line) and push the numbers into S one by one
142 |     printf("1. Use rand() to get 10 random numbers, output the numbers (one by one, one space in between, and 8 numbers in one line) and push the numbers into S one by one\n");
143 |     random_number_generator(&s, 100, 1, 10);
144 |     printf("1. // 10 numbers per line\n");
145 |     printf("un\n");
146 |
147 |     // 2. Assign and output integer x the 11th element from the top of S, leaving S unchanged.
148 |     printf("2. Assign and output integer x the 11th element from the top of S, leaving S unchanged.\n");
149 |     int x = get_element(&s, FROM_TOP, 11, RECOVER);
150 |     printf("x = %d\n", x);
151 |     printf("un\n");
152 |
153 |     // 3. Put integer n in (2) under the bottom of S (leaving the rest of S unchanged) and output the numbers (one by one, one space in between, and 8 numbers in one line) on S from the top to the bottom.
154 |     reverse_stack(&s);
155 | }
```

1.4 Bracket Matching

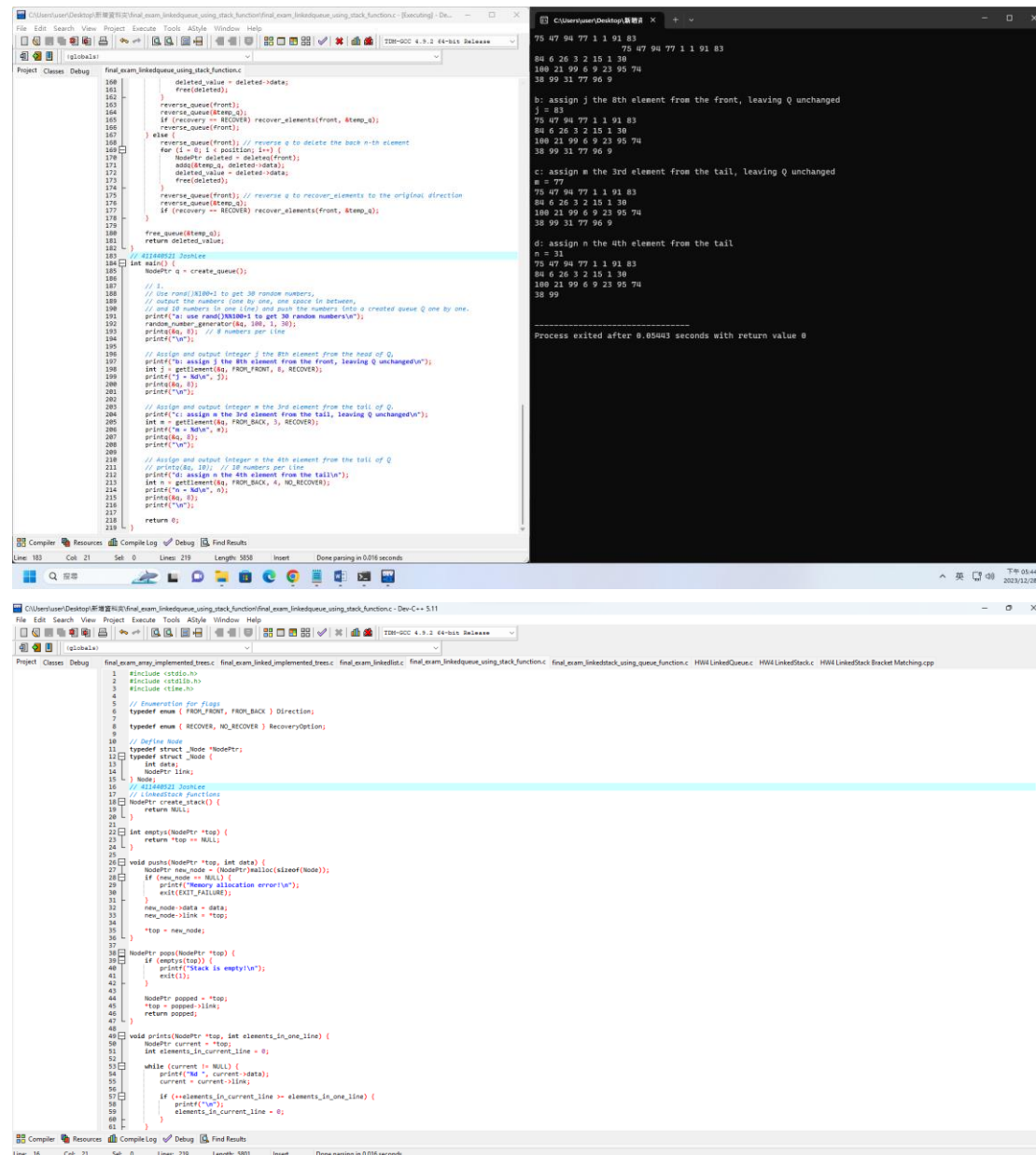
```
28 {
29     BracketPtr popped = *top;
30     *top = popped->link;
31     return popped;
32 }
33
34 // Create a new LinkedStack
35 BracketPtr create_stack() { return NULL; }
36
37 // Self-defined
38 int is_left(char symb) { return symb == '(' || symb == '[' || symb == '{'; }
39 int is_right(char symb) { return symb == ')' || symb == ']' || symb == '}'; }
40
41 char correspond(char type)
42 {
43     if (type == '(') return ')';
44     if (type == '[') return ']';
45     if (type == '{') return '}';
46     if (type == ')') return '(';
47     if (type == ']') return '[';
48     if (type == '}') return '{';
49     return '\0';
50 }
51
52 // 41440922 Jashlee
53 int main()
54 {
55     BracketPtr top = create_stack();
56
57     char symb;
58     int pos = 0;
59
60     while ((symb = getchar()) != '\n') {
61         if (is_left(symb)) {
62             push(top, symb, pos);
63         } else if (is_right(symb)) {
64             if (empty(top)) {
65                 BracketPtr left = pop(top);
66                 if (symb == correspond(left->symb)) {
67                     printf("No Mismatch, left: %s, pos: %d, symb: %c", symb, pos, symb);
68                     free(left); // free the memory allocated for the popped node
69                 } else {
69                     printf("right parenthesis %c at %d has no matching left parenthesis %c\n", symb, pos, correspond(symb));
70                     free(left); // free the memory allocated for the popped node
71                 }
72             } else {
73                 printf("right parenthesis %c at %d has no matching left parenthesis %c\n", symb, pos, correspond(symb));
74                 pos++;
75             }
76         }
77     }
78
79     while (!empty(top)) {
80         BracketPtr left = pop(top);
81         printf("left parenthesis %c at %d has no matching right parenthesis %c\n", left->symb, left->pos, correspond(left->symb));
82         free(left); // free the memory allocated for the popped node
83     }
84
85     return 0;
86 }
```

```
(((c+b)))f(((c+d)))
(3,7)
right parenthesis ] at 8 has no matching left parenthesis (
right parenthesis ) at 16 has no matching left parenthesis (
[15,17]
left parenthesis [ at 10 has no matching right parenthesis ]
left parenthesis ( at 1 has no matching right parenthesis )
left parenthesis ( at 0 has no matching right parenthesis )

Process exited after 23.43 seconds with return value 0
請按任意鍵繼續 . . .
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 // 41440922 Jashlee
4 // Define Bracket
5 typedef struct _Bracket {
6     char symb;
7     int pos;
8     BracketPtr link;
9 } Bracket;
10
11 // LinkedStack Functions
12 int empty(BracketPtr *top) { return *top == NULL; }
13
14 void push(BracketPtr *top, char symb, int pos) {
15     BracketPtr new_node = (Bracket *)malloc(sizeof(Bracket));
16     new_node->symb = symb;
17     new_node->pos = pos;
18     new_node->link = *top;
19     *top = new_node;
20 }
21
22 BracketPtr pop(BracketPtr *top) {
23     if (empty(top)) {
24         printf("Stack is empty!\n");
25         exit(1);
26     }
27     BracketPtr popped = *top;
28     *top = popped->link;
29     return popped;
30 }
31
32 // Create a new LinkedStack
33 BracketPtr create_stack() { return NULL; }
34
35 // Self-defined
36 int is_left(char symb) { return symb == '(' || symb == '[' || symb == '{'; }
37 int is_right(char symb) { return symb == ')' || symb == ']' || symb == '}'; }
38
39 char correspond(char type)
40 {
41     if (type == '(') return ')';
42     if (type == '[') return ']';
43     if (type == '{') return '}';
44     if (type == ')') return '(';
45     if (type == ']') return '[';
46     if (type == '}') return '{';
47     return '\0';
48 }
49
50 // 41440922 Jashlee
51 int main()
52 {
53     BracketPtr top = create_stack();
54
55     char symb;
56     int pos = 0;
57
58     while ((symb = getchar()) != '\n') {
59         if (is_left(symb)) {
```

2.1 Linked Queues by Linked Stacks



```
160 // deleted value = deleted data;
161 // free(deleted);
162 // free(deleted);
163 reverse_queue(front);
164 reverse_queue(&temp_q);
165 if (recovery == RECOVER) recover_elements(front, &temp_q);
166 reverse_queue(front);
167 } else {
168 // reverse queue to delete the back n-th element
169 for (i = 0; i < position; i++) {
170 // NodePtr deleted = delete(front);
171 // add(deleted, deleted_data);
172 // deleted_value = deleted_data;
173 // free(deleted);
174 // reverse_queue(front); // reverse q to recover elements to the original direction
175 // reverse_queue(&temp_q);
176 // if (recovery == RECOVER) recover_elements(front, &temp_q);
177 }
178 // free_queue(&temp_q);
179 return deleted_value;
180 }
181 // if (isQueueEmpty) {
182 // NodePtr q = create_queue();
183 // }
184 // // Use rand() to get 30 random numbers;
185 // // output the numbers (one by one, one space in between);
186 // // add numbers to one line and push the numbers into a created queue q one by one.
187 // print("a: use rand() to get 30 random numbers\n");
188 // random_number_generator(10, 100, 1, 30);
189 // print(q); // 0 numbers per line
190 // print("\n");
191 // Assign and output integer j the 8th element from the head of Q;
192 // print("b: assign j the 8th element from the head, leaving Q unchanged\n");
193 // int j = get_element(q, FROM_FRONT, 8, RECOVER);
194 // print("j = %d\n", j);
195 // print(q);
196 // Assign and output integer m the 3rd element from the tail of Q;
197 // print("c: assign m the 3rd element from the tail, leaving Q unchanged\n");
198 // int m = get_element(q, FROM_BACK, 3, RECOVER);
199 // print("m = %d\n", m);
200 // print(q);
201 // Assign and output integer n the 4th element from the tail of Q;
202 // print("d: assign n the 4th element from the tail\n");
203 // int n = get_element(q, FROM_BACK, 4, NO_RECOVER);
204 // print("n = %d\n", n);
205 // print(q);
206 // return 0;
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
```

```
75 47 94 77 1 1 91 83
84 6 26 3 2 15 1 30
100 21 99 6 9 23 95 74
38 99 31 77 96 9

b: assign j the 8th element from the front, leaving Q unchanged
j = 83
75 47 94 77 1 1 91 83
84 6 26 3 2 15 1 30
100 21 99 6 9 23 95 74
38 99 31 77 96 9

c: assign m the 3rd element from the tail, leaving Q unchanged
m = 77
75 47 94 77 1 1 91 83
84 6 26 3 2 15 1 30
100 21 99 6 9 23 95 74
38 99 31 77 96 9

d: assign n the 4th element from the tail
n = 31
75 47 94 77 1 1 91 83
84 6 26 3 2 15 1 30
100 21 99 6 9 23 95 74
38 99 31 77 96 9

Process exited after 0.05043 seconds with return value 0
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 // Enumeration for flags
6 typedef enum { FROM_FRONT, FROM_BACK } Direction;
7
8 typedef enum { RECOVER, NO_RECOVER } RecoveryOption;
9
10 // Define Node
11 typedef struct _NodePtr {
12     int data;
13     NodePtr link;
14 } Node;
15
16 // LinkedStack functions
17 // LinkedStack functions
18 // NodePtr create_stack() {
19 //     return NULL;
20 // }
21 // int empty(NodePtr *top) {
22 //     return *top == NULL;
23 // }
24 // void push(NodePtr *top, int data) {
25 //     NodePtr new_node = (NodePtr)malloc(sizeof(Node));
26 //     if (new_node == NULL) {
27 //         printf("Memory allocation error\n");
28 //         exit(EXIT_FAILURE);
29 //     }
30 //     new_node->data = data;
31 //     new_node->link = *top;
32 //     *top = new_node;
33 // }
34 // NodePtr pop(NodePtr *top) {
35 //     if (empty(top)) {
36 //         printf("Stack is empty\n");
37 //         exit(1);
38 //     }
39 //     NodePtr popped = *top;
40 //     *top = popped->link;
41 //     return popped;
42 // }
43 // void prints(NodePtr *top, int elements_in_one_line) {
44 //     NodePtr current = *top;
45 //     int elements_in_current_line = 0;
46 //     while (current != NULL) {
47 //         printf("%d ", current->data);
48 //         current = current->link;
49 //         if (++elements_in_current_line == elements_in_one_line) {
50 //             printf("\n");
51 //             elements_in_current_line = 0;
52 //         }
53 //     }
54 // }
```

```

48 void print(NodePtr *top, int elements_in_one_line) {
49     NodePtr current = *top;
50     int elements_in_current_line = 0;
51     while (current != NULL) {
52         printf("%d ", current->data);
53         current = current->next;
54         if (++elements_in_current_line == elements_in_one_line) {
55             printf("\n");
56             elements_in_current_line = 0;
57         }
58     }
59     printf(elements_in_current_line > "0\n" : "");
60 }
61
62 void free_stack(NodePtr *s) {
63     while (!empty(s)) {
64         NodePtr popped = pop(s);
65         free(popped);
66     }
67 }
68
69 // 41144052 Jachies
70 // LinkedQueue Functions (Using LinkedStack Functions)
71
72 NodePtr create_queue() {
73     return create_stack();
74 }
75
76 int empty(NodePtr *front) {
77     return empty(front);
78 }
79
80 void add(NodePtr *front, int data) {
81     NodePtr temp_s = create_stack();
82     // Reverse the order of elements in the stack
83     while (!empty(front)) {
84         NodePtr popped = pop(front);
85         push(temp_s, popped->data);
86         free(popped);
87     }
88     // Add the new element at the top
89     push(front, data);
90     // Restore the original order of elements
91     while (!empty(temp_s)) {
92         NodePtr popped = pop(temp_s);
93         push(front, popped->data);
94         free(popped);
95     }
96 }
97
98 NodePtr deleted(NodePtr *front) {
99     if (empty(front)) {
100         printf("Queue is empty!\n");
101         exit(1);
102     }
103     return pop(front);
104 }
105
106 // 41144052 Jachies
107 // Question
108 void random_number_generator(NodePtr *front, int range, int offset, int len) {
109     srand(time(NULL));
110     while (!empty(front)) deleted(front);
111     int i;
112     for (i = 0; i < len; i++) add(front, rand() % range + offset);
113 }
114
115 // Helper function to reverse the queue
116 void reverse_queue(NodePtr *q) {
117     if (empty(q)) {
118         return;
119     }
120     int fr = deleted(q)->data;
121     reverse_queue(q);
122     add(q, fr);
123 }
124
125 // Helper function to recover elements to the original stack
126 void recover_elements(NodePtr *q, NodePtr *temp_a) {
127     while (!empty(temp_a)) {
128         NodePtr deleted = deleted(temp_a);
129         add(q, deleted->data);
130         free(deleted);
131     }
132 }
133
134 // Helper function to get an element from the stack
135 int get_element(NodePtr *front, direction direction, int position, RecoveryOption recovery) {
136     NodePtr temp_a = create_queue();
137     int i, deleted_value;
138     if (direction == FROM_FRONT) {
139         for (i = 0; i < position; i++) {
140             NodePtr deleted = deleted(front);
141             add(temp_a, deleted->data);
142             deleted_value = deleted->data;
143             free(deleted);
144         }
145         reverse_queue(front);
146     }
147     if (recovery == RECOVER) recover_elements(front, temp_a);
148     reverse_queue(front);
149     if (recovery == RECOVER) recover_elements(front, temp_a);
150     if (i > 0) {
151         NodePtr deleted = deleted(temp_a);
152         add(temp_a, deleted->data);
153         deleted_value = deleted->data;
154         free(deleted);
155     }
156     return deleted_value;
157 }
158
159 // 41144052 Jachies
160 int main() {
161     NodePtr q = create_queue();
162     // 1.
163     // Use rand() % 10 to get 10 random numbers.
164     // Using the numbers, use pop, one space in between,
165     // and 10 numbers in one line and push the numbers into a created queue q one by one.
166     printf("I use rand() % 10 to get 10 random numbers.\n");
167     random_number_generator(&q, 10, 1, 10);
168     printf("%d", q); // 0 number per line
169 }
```

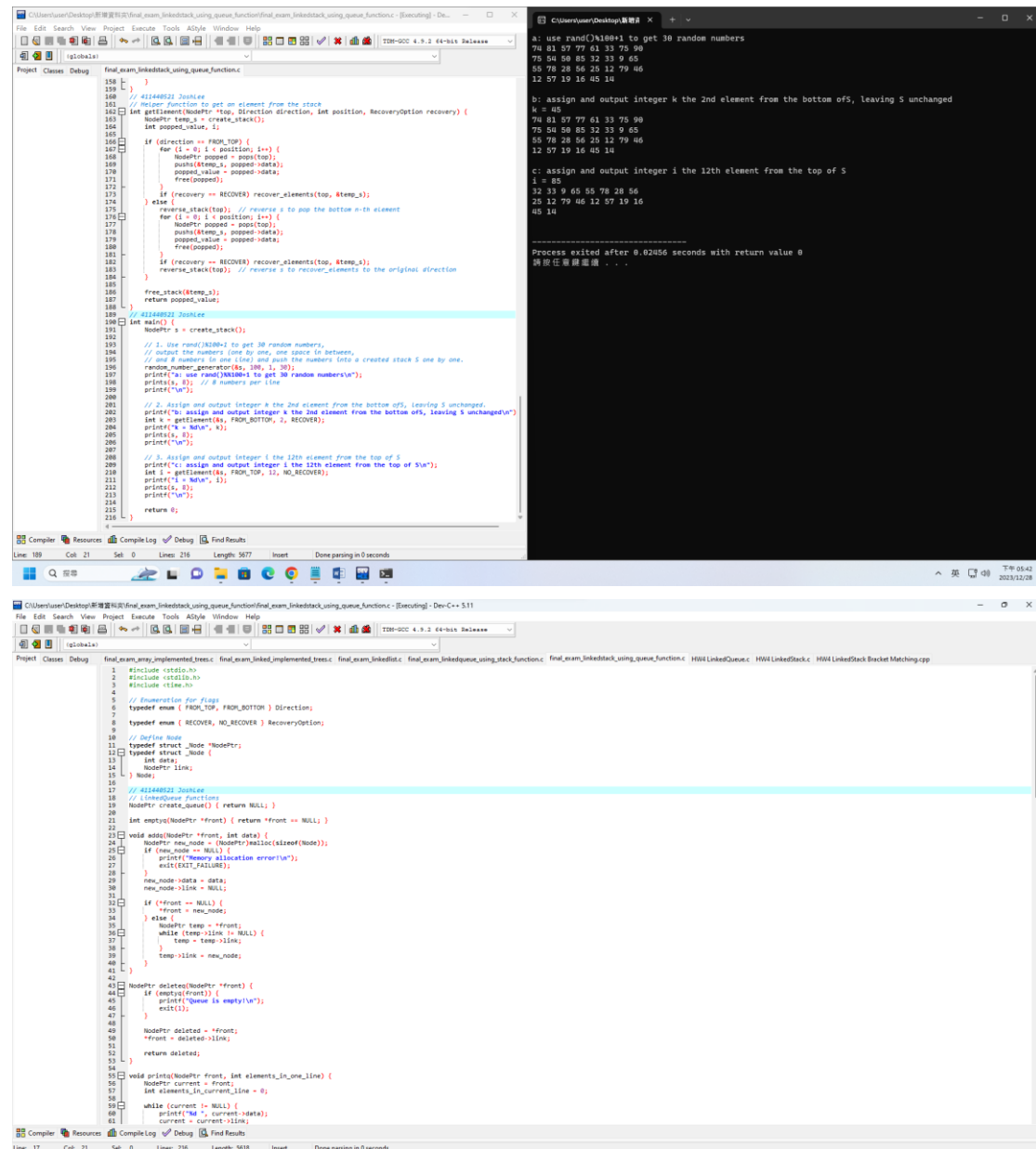
```

169 NodePtr deleted(NodePtr *front) {
170     if (empty(front)) {
171         printf("Queue is empty!\n");
172         exit(1);
173     }
174     return pop(front);
175 }
176
177 void print(NodePtr *front, int elements_in_one_line) {
178     print(front, elements_in_one_line);
179 }
180
181 void free_queue(NodePtr *front) {
182     free_stack(front);
183 }
184
185 // 41144052 Jachies
186 // Question
187 void random_number_generator(NodePtr *front, int range, int offset, int len) {
188     srand(time(NULL));
189     while (!empty(front)) deleted(front);
190     int i;
191     for (i = 0; i < len; i++) add(front, rand() % range + offset);
192 }
193
194 // Helper function to reverse the queue
195 void reverse_queue(NodePtr *q) {
196     if (empty(q)) {
197         return;
198     }
199     int fr = deleted(q)->data;
200     reverse_queue(q);
201     add(q, fr);
202 }
203
204 // Helper function to recover elements to the original stack
205 void recover_elements(NodePtr *q, NodePtr *temp_a) {
206     while (!empty(temp_a)) {
207         NodePtr deleted = deleted(temp_a);
208         add(q, deleted->data);
209         free(deleted);
210     }
211 }
212
213 // Helper function to get an element from the stack
214 int get_element(NodePtr *front, direction direction, int position, RecoveryOption recovery) {
215     NodePtr temp_a = create_queue();
216     int i, deleted_value;
217     if (direction == FROM_FRONT) {
218         for (i = 0; i < position; i++) {
219             NodePtr deleted = deleted(front);
220             add(temp_a, deleted->data);
221             deleted_value = deleted->data;
222             free(deleted);
223         }
224         reverse_queue(front);
225     }
226     if (recovery == RECOVER) recover_elements(front, temp_a);
227     reverse_queue(front);
228     if (recovery == RECOVER) recover_elements(front, temp_a);
229     if (i > 0) {
230         NodePtr deleted = deleted(temp_a);
231         add(temp_a, deleted->data);
232         deleted_value = deleted->data;
233         free(deleted);
234     }
235     return deleted_value;
236 }
237
238 // 41144052 Jachies
239 int main() {
240     NodePtr q = create_queue();
241     // 1.
242     // Use rand() % 10 to get 10 random numbers.
243     // Using the numbers, use pop, one space in between,
244     // and 10 numbers in one line and push the numbers into a created queue q one by one.
245     printf("I use rand() % 10 to get 10 random numbers.\n");
246     random_number_generator(&q, 10, 1, 10);
247     printf("%d", q); // 0 number per line
248 }
```

```

249 }
250
251 int fr = deleted(q)->data;
252 reverse_queue(q);
253 add(q, fr);
254 }
255
256 // Helper function to recover elements to the original stack
257 void recover_elements(NodePtr *q, NodePtr *temp_a) {
258     while (!empty(temp_a)) {
259         NodePtr deleted = deleted(temp_a);
260         add(q, deleted->data);
261         free(deleted);
262     }
263 }
264
265 // 41144052 Jachies
266 // Helper function to get an element from the stack
267 int get_element(NodePtr *front, direction direction, int position, RecoveryOption recovery) {
268     NodePtr temp_a = create_queue();
269     int i, deleted_value;
270     if (direction == FROM_FRONT) {
271         for (i = 0; i < position; i++) {
272             NodePtr deleted = deleted(front);
273             add(temp_a, deleted->data);
274             deleted_value = deleted->data;
275             free(deleted);
276         }
277         reverse_queue(front);
278     }
279     if (recovery == RECOVER) recover_elements(front, temp_a);
280     reverse_queue(front);
281     if (recovery == RECOVER) recover_elements(front, temp_a);
282     if (i > 0) {
283         NodePtr deleted = deleted(temp_a);
284         add(temp_a, deleted->data);
285         deleted_value = deleted->data;
286         free(deleted);
287     }
288     reverse_queue(front); // reverse q to recover elements to the original direction
289     if (recovery == RECOVER) recover_elements(front, temp_a);
290     if (i > 0) {
291         NodePtr deleted = deleted(temp_a);
292         add(temp_a, deleted->data);
293         deleted_value = deleted->data;
294         free(deleted);
295     }
296     return deleted_value;
297 }
298
299 // 41144052 Jachies
300 int main() {
301     NodePtr q = create_queue();
302     // 1.
303     // Use rand() % 10 to get 10 random numbers.
304     // Using the numbers, use pop, one space in between,
305     // and 10 numbers in one line and push the numbers into a created queue q one by one.
306     printf("I use rand() % 10 to get 10 random numbers.\n");
307     random_number_generator(&q, 10, 1, 10);
308     printf("%d", q); // 0 number per line
309 }
```

2.2 Linked Stacks by Linked Queues



```
C:\Users\user\Desktop\新舊資料夾\final_exam_linkedstack_using_queue_function\final_exam_linkedstack_using_queue_function.c - [Encoding: Dev-C++ 5.11]
File Edit Search View Project Execute Tools Alpha Window Help
TDS-GCC 4.9.2 64-bit Release
Project Classes Debug
final_exam_arry_implemented_btree final_exam_linked_implemented_btree final_exam_linkedlist.c final_exam_linkedqueue_using_stack_function.c final_exam_linkedstack_using_queue_function.c H001LinkedQueue.c H001LinkedStack.c H001LinkedStack Bracket Matching.cpp
33 void print(NodePtr front, int elements_in_one_line) {
34     NodePtr current = front;
35     int elements_in_current_line = 0;
36     while (current != NULL) {
37         printf("%d ", current->data);
38         current = current->link;
39         if (++elements_in_current_line >= elements_in_one_line) {
40             printf("\n");
41             elements_in_current_line = 0;
42         }
43     }
44     printf(elements_in_current_line > "0\n" : "");
45 }
46 // Helper function to reverse the queue
47 void reverse_queue(NodePtr *q) {
48     if (empty(q)) {
49         return;
50     }
51     int fr = delete(q->data);
52     reverse_queue(q);
53     add(q, fr);
54 }
55 void free_queue(NodePtr *q) {
56     while (!empty(q)) {
57         free(delete(q));
58     }
59 }
60 // 41144052 Jachine
61 // LinkedStack functions (using LinkedQueue Functions)
62 NodePtr create_stack() {
63     return create_queue();
64 }
65 int empty(NodePtr *top) {
66     return empty(top);
67 }
68 void push(NodePtr *top, int data) {
69     // Push all elements to temp_q
70     NodePtr temp_q = create_queue();
71     while (!empty(top)) {
72         NodePtr deleted = delete(top);
73         add(temp_q, deleted->data);
74         free(deleted);
75     }
76     // Add the new element to the top of the stack
77     add(top, data);
78     // Recover elements back to the original stack
79     while (!empty(temp_q)) {
80         NodePtr deleted = delete(temp_q);
81         add(top, deleted->data);
82     }
83 }
84 void pop(NodePtr *top) {
85     if (empty(top)) {
86         printf("Stack is empty!\n");
87         exit(EXIT_FAILURE);
88     }
89     return delete(top);
90 }
91 void reverse_stack(NodePtr *top) {
92     reverse_queue(top);
93 }
94 void print(NodePtr top, int elements_in_one_line) {
95     print(top, elements_in_one_line);
96 }
97 void free_stack(NodePtr *top) {
98     free_queue(top);
99 }
100 // 41144052 Jachine
101 // Self-Defined
102 void random_number_generator(NodePtr *front, int range, int offset, int len) {
103     srand(time(NULL));
104     while (!empty(front)) delete(front);
105     int i;
106     for (i = 0; i < len; i++) add(front, rand() % range + offset);
107 }
108 // Helper function to recover elements to the original stack
109 void recover_elements(NodePtr *top, NodePtr *temp_q) {
110     while (!empty(temp_q)) {
111         NodePtr popped = pop(temp_q);
112         push(top, popped->data);
113         free(popped);
114     }
115 }
116 // 41144052 Jachine
117 // Helper function to get an element from the stack
118 int get_element(NodePtr *top, Direction direction, int position, RecoveryOption recovery) {
119     NodePtr temp_s = create_stack();
120     int popped_value = 0;
121     if (direction == FROM_TOP) {
122         for (i = 0; i < position; i++) {
123             NodePtr popped = pop(top);
124             push(temp_s, popped->data);
125             popped_value = popped->data;
126             free(popped);
127         }
128         if (recovery == RECOVER) recover_elements(top, temp_s);
129     } else {
130         reverse_stack(top); // reverse s to pop the bottom n-th element
131         for (i = 0; i < position; i++) {
132             NodePtr popped = pop(top);
133             push(temp_s, popped->data);
134             popped_value = popped->data;
135             free(popped);
136         }
137         if (recovery == RECOVER) recover_elements(top, temp_s);
138     }
139     reverse_stack(top); // reverse s to recover elements to the original direction
140     free_stack(temp_s);
141     return popped_value;
142 }
143 // 41144052 Jachine
144 int main() {
145     NodePtr s = create_stack();
146     // 1. Use rand() to get 30 random numbers.
147     // Output the numbers line by line, one value in between.
148     // Use 5 numbers in one line and push the numbers into a created stack s one by one.
149     random_number_generator(&s, 100, 1, 30);
150     printf("1. Use rand() to get 30 random numbers\n");
151     print(s, 5);
152     // 2. Assign and output integer n the 2nd element from the bottom of s, leaving s unchanged.
153     print("2. Assign and output integer n the 2nd element from the bottom of s, leaving s unchanged\n");
154     int n = get_element(s, FROM_BOTTOM, 2, RECOVER);
155     printf("n = %d\n", n);
156     print(s, 5);
157 }
```

```
C:\Users\user\Desktop\新舊資料夾\final_exam_linkedstack_using_queue_function\final_exam_linkedstack_using_queue_function.c - [Encoding: Dev-C++ 5.11]
File Edit Search View Project Execute Tools Alpha Window Help
TDS-GCC 4.9.2 64-bit Release
Project Classes Debug
final_exam_arry_implemented_btree final_exam_linked_implemented_btree final_exam_linkedlist.c final_exam_linkedqueue_using_stack_function.c final_exam_linkedstack_using_queue_function.c H001LinkedQueue.c H001LinkedStack.c H001LinkedStack Bracket Matching.cpp
100 void push(NodePtr *top, int data) {
101     // Push all elements to temp_q
102     NodePtr temp_q = create_queue();
103     while (!empty(top)) {
104         NodePtr deleted = delete(top);
105         add(temp_q, deleted->data);
106         free(deleted);
107     }
108     // Add the new element to the top of the stack
109     add(top, data);
110     // Recover elements back to the original stack
111     while (!empty(temp_q)) {
112         NodePtr deleted = delete(temp_q);
113         add(top, deleted->data);
114         free(deleted);
115     }
116 }
117 void pop(NodePtr *top) {
118     if (empty(top)) {
119         printf("Stack is empty!\n");
120         exit(EXIT_FAILURE);
121     }
122     return delete(top);
123 }
124 void reverse_stack(NodePtr *top) {
125     reverse_queue(top);
126 }
127 void print(NodePtr top, int elements_in_one_line) {
128     print(top, elements_in_one_line);
129 }
130 void free_stack(NodePtr *top) {
131     free_queue(top);
132 }
133 // 41144052 Jachine
134 // Self-Defined
135 void random_number_generator(NodePtr *front, int range, int offset, int len) {
136     srand(time(NULL));
137     while (!empty(front)) delete(front);
138     int i;
139     for (i = 0; i < len; i++) add(front, rand() % range + offset);
140 }
141 // Helper function to recover elements to the original stack
142 void recover_elements(NodePtr *top, NodePtr *temp_q) {
143     while (!empty(temp_q)) {
144         NodePtr popped = pop(temp_q);
145         push(top, popped->data);
146         free(popped);
147     }
148 }
149 // 41144052 Jachine
150 // Helper function to get an element from the stack
151 int get_element(NodePtr *top, Direction direction, int position, RecoveryOption recovery) {
152     NodePtr temp_s = create_stack();
153     int popped_value = 0;
154     if (direction == FROM_TOP) {
155         for (i = 0; i < position; i++) {
156             NodePtr popped = pop(top);
157             push(temp_s, popped->data);
158             popped_value = popped->data;
159             free(popped);
160         }
161         if (recovery == RECOVER) recover_elements(top, temp_s);
162     } else {
163         reverse_stack(top); // reverse s to pop the bottom n-th element
164         for (i = 0; i < position; i++) {
165             NodePtr popped = pop(top);
166             push(temp_s, popped->data);
167             popped_value = popped->data;
168             free(popped);
169         }
170         if (recovery == RECOVER) recover_elements(top, temp_s);
171     }
172     reverse_stack(top); // reverse s to recover elements to the original direction
173     free_stack(temp_s);
174     return popped_value;
175 }
176 // 41144052 Jachine
177 int main() {
178     NodePtr s = create_stack();
179     // 1. Use rand() to get 30 random numbers.
180     // Output the numbers line by line, one value in between.
181     // Use 5 numbers in one line and push the numbers into a created stack s one by one.
182     random_number_generator(&s, 100, 1, 30);
183     printf("1. Use rand() to get 30 random numbers\n");
184     print(s, 5);
185     // 2. Assign and output integer n the 2nd element from the bottom of s, leaving s unchanged.
186     print("2. Assign and output integer n the 2nd element from the bottom of s, leaving s unchanged\n");
187     int n = get_element(s, FROM_BOTTOM, 2, RECOVER);
188     printf("n = %d\n", n);
189     print(s, 5);
190 }
```

```
C:\Users\user\Desktop\新舊資料夾\final_exam_linkedstack_using_queue_function\final_exam_linkedstack_using_queue_function.c - [Encoding: Dev-C++ 5.11]
File Edit Search View Project Execute Tools Alpha Window Help
TDS-GCC 4.9.2 64-bit Release
Project Classes Debug
final_exam_arry_implemented_btree final_exam_linked_implemented_btree final_exam_linkedlist.c final_exam_linkedqueue_using_stack_function.c final_exam_linkedstack_using_queue_function.c H001LinkedQueue.c H001LinkedStack.c H001LinkedStack Bracket Matching.cpp
140 void random_number_generator(NodePtr *front, int range, int offset, int len) {
141     srand(time(NULL));
142     while (!empty(front)) delete(front);
143     int i;
144     for (i = 0; i < len; i++) add(front, rand() % range + offset);
145 }
146 // Helper function to recover elements to the original stack
147 void recover_elements(NodePtr *top, NodePtr *temp_q) {
148     while (!empty(temp_q)) {
149         NodePtr popped = pop(temp_q);
150         push(top, popped->data);
151         free(popped);
152     }
153 }
154 // 41144052 Jachine
155 // Helper function to get an element from the stack
156 int get_element(NodePtr *top, Direction direction, int position, RecoveryOption recovery) {
157     NodePtr temp_s = create_stack();
158     int popped_value = 0;
159     if (direction == FROM_TOP) {
160         for (i = 0; i < position; i++) {
161             NodePtr popped = pop(top);
162             push(temp_s, popped->data);
163             popped_value = popped->data;
164             free(popped);
165         }
166         if (recovery == RECOVER) recover_elements(top, temp_s);
167     } else {
168         reverse_stack(top); // reverse s to pop the bottom n-th element
169         for (i = 0; i < position; i++) {
170             NodePtr popped = pop(top);
171             push(temp_s, popped->data);
172             popped_value = popped->data;
173             free(popped);
174         }
175         if (recovery == RECOVER) recover_elements(top, temp_s);
176     }
177     reverse_stack(top); // reverse s to recover elements to the original direction
178     free_stack(temp_s);
179     return popped_value;
180 }
181 // 41144052 Jachine
182 int main() {
183     NodePtr s = create_stack();
184     // 1. Use rand() to get 30 random numbers.
185     // Output the numbers line by line, one value in between.
186     // Use 5 numbers in one line and push the numbers into a created stack s one by one.
187     random_number_generator(&s, 100, 1, 30);
188     printf("1. Use rand() to get 30 random numbers\n");
189     print(s, 5);
190     // 2. Assign and output integer n the 2nd element from the bottom of s, leaving s unchanged.
191     print("2. Assign and output integer n the 2nd element from the bottom of s, leaving s unchanged\n");
192     int n = get_element(s, FROM_BOTTOM, 2, RECOVER);
193     printf("n = %d\n", n);
194     print(s, 5);
195 }
```


[illegible]

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 // Define list Node
6 typedef struct Node {
7     int data; // Store value of the node
8     NodePtr next; // Store address of the next node
9 } Node;
10
11 // #11440523 Jonhlee
12
13 void insert(NodePtr *target, int data) { // double pointer for adding first element
14     NodePtr new_node = (NodePtr)malloc(sizeof(Node));
15     new_node->data = data;
16     new_node->next = NULL;
17
18     // For empty list
19     if (*target == NULL) {
20         *target = new_node;
21         return;
22     }
23
24     // For non-empty list
25     new_node->next = (*target)->next;
26     (*target)->next = new_node;
27 }
28
29 int delete(NodePtr *target) { // double pointer for deleting last element
30     // For empty
31     if (*target == NULL) {
32         printf("List is empty");
33         exit(1);
34     }
35
36     // For the last element
37     if ((*target)->next == NULL) { // For normal list
38         (*target)->next = *target; // For circular list
39         NodePtr deleted = *target;
40         int data = deleted->data;
41         *target = NULL;
42         free(deleted);
43         return data;
44     }
45
46     // For other elements
47     NodePtr deleted = (*target)->next;
48     int data = deleted->data;
49     (*target)->next = deleted->next;
50     free(deleted);
51     return data;
52 }
53
54 // #11440523 Jonhlee
55 NodePtr insertnum_20(NodePtr *head, int value) {
56     NodePtr *current = head; // Use an indirect pointer
57
58     // Traverse the list to find the appropriate position
59     while (*current != NULL && (*current)->data < value) {
60         current = &(*current)->next;
61     }
62
63     // Create a new node and insert it into the list
64     NodePtr new_node = (NodePtr)malloc(sizeof(Node));
65     new_node->data = value;
66     new_node->next = *current;
67     *current = new_node;
68
69     return new_node;
70 }
71
72 NodePtr insertnum_21(NodePtr *head, int value) {
73     NodePtr *current = head; // Use an indirect pointer
74
75     // Traverse the list to find the appropriate position
76     while (*current != NULL && (*current)->data < value) {
77         current = &(*current)->next;
78     }
79
80     // Create a new node and insert it into the list
81     NodePtr new_node = (NodePtr)malloc(sizeof(Node));
82     new_node->data = value;
83     new_node->next = *current;
84     *current = new_node;
85
86     return new_node;
87 }
88
89 NodePtr concatenate(NodePtr p, NodePtr q) {
90     if (p == NULL)
91         return q;
92
93     NodePtr current = p;
94     while (current->next != NULL)
95         current = current->next;
96     current->next = q;
97
98     return p;
99 }
100
101 NodePtr invert(NodePtr head) {
102     NodePtr prev = NULL;
103     NodePtr current = head;
104     NodePtr next_node;
105
106     while (current != NULL) {
107         next_node = current->next;
108         current->next = prev;
109         prev = current;
110         current = next_node;
111     }
112
113     return prev;
114 }
115
116 // #11440523 Jonhlee
```

```
55 // #11440523 Jonhlee
56 NodePtr insertnum_20(NodePtr *head, int value) {
57     NodePtr *current = head; // Use an indirect pointer
58
59     // Traverse the list to find the appropriate position
60     while (*current != NULL && (*current)->data < value) {
61         current = &(*current)->next;
62     }
63
64     // Create a new node and insert it into the list
65     NodePtr new_node = (NodePtr)malloc(sizeof(Node));
66     new_node->data = value;
67     new_node->next = *current;
68     *current = new_node;
69
70     return new_node;
71 }
72
73 NodePtr insertnum_21(NodePtr *head, int value) {
74     NodePtr *current = head; // Use an indirect pointer
75
76     // Traverse the list to find the appropriate position
77     while (*current != NULL && (*current)->data < value) {
78         current = &(*current)->next;
79     }
80
81     // Create a new node and insert it into the list
82     NodePtr new_node = (NodePtr)malloc(sizeof(Node));
83     new_node->data = value;
84     new_node->next = *current;
85     *current = new_node;
86
87     return new_node;
88 }
89
90 NodePtr concatenate(NodePtr p, NodePtr q) {
91     if (p == NULL)
92         return q;
93
94     NodePtr current = p;
95     while (current->next != NULL)
96         current = current->next;
97     current->next = q;
98
99     return p;
100 }
101
102 NodePtr invert(NodePtr head) {
103     NodePtr prev = NULL;
104     NodePtr current = head;
105     NodePtr next_node;
106
107     while (current != NULL) {
108         next_node = current->next;
109         current->next = prev;
110         prev = current;
111         current = next_node;
112     }
113
114     return prev;
115 }
116
117 // #11440523 Jonhlee
```

```
113
114
115 // #11440523 Jonhlee
116
117 NodePtr listCirc(NodePtr head) {
118     NodePtr current = head;
119
120     // get last node
121     while (current->next != NULL)
122         current = current->next;
123
124     // last -> head
125     current->next = head;
126
127     return head;
128 }
129
130 void print(NodePtr head) {
131     NodePtr current = head;
132     int elements_in_current_line = 0;
133     int elements_in_one_line = 0;
134
135     while (current != NULL) {
136         printf("%d ", current->data);
137         current = current->next;
138
139         if (++elements_in_current_line >= elements_in_one_line) {
140             printf("\n");
141             elements_in_current_line = 0;
142         }
143     }
144     printf(elements_in_current_line > "0" ? "\n" : "");
145 }
146
147 void print(NodePtr head) {
148     // Empty list
149     if (head == NULL) {
150         return;
151     }
152
153     int elements_in_one_line = 0;
154     int elements_in_current_line = 0;
155     NodePtr current = head;
156
157     do {
158         printf("%d ", current->data);
159         current = current->next;
160
161         if (++elements_in_current_line >= elements_in_one_line) {
162             printf("\n");
163             elements_in_current_line = 0;
164         }
165     } while (current != head);
166     printf(elements_in_current_line > "0" ? "\n" : "");
167 }
168
169 NodePtr create_list() { return NULL; }
170
171 // #11440523 Jonhlee
```

```
C:\Users\user\Desktop\新管理架构\final_exam_linkedlist\final_exam_linkedlist - Dev-C++ 5.11
File Edit Search View Project RunTime Tools Alpha Window Help
TDS-OC 4.9.2 64-bit Release
Project Classes Debug
final_exam_array_implemented_test.c final_exam_linked_implemented_test.c final_exam_linkedlist.c final_exam_linkedqueue_using_stack_function.c final_exam_linkedstack_using_queue_function.c HWLinkedList.c HWLinkedList.c HWLinkedListBracketMatching.cpp

212 // 411648021 JoshuaLee
213 //
214 // Extensions
215 //
216 //
217 void append(NodePtr *target, int data) {
218     // Traverse the list until the end is reached
219     while (*target != NULL)
220         // Move to the next node using its next pointer
221         target = &(*target->next);
222     NodePtr new_node = (NodePtr)malloc(sizeof(Node));
223     new_node->data = data;
224     new_node->next = NULL;
225     *target = new_node;
226 }
227
228 NodePtr advance(NodePtr current, int steps) {
229     int i;
230     for (i = 0; i < steps && current != NULL; i++)
231         current = current->next;
232     return current;
233 }
234
235 void free_list(NodePtr *head) {
236     // Current list node, used for traversing the list
237     NodePtr current = *head;
238     while (current != NULL) {
239         NodePtr temp = current; // Temporarily store the current node
240         current = current->next; // Move to the next node
241         free(temp); // Free the memory of the node
242     }
243     *head = NULL; // Set the external pointer (list-head) to NULL
244 }
245
246 NodePtr filter(int (*condition)(int), NodePtr head) {
247     NodePtr result = create_list();
248     NodePtr current = head;
249     while (current != NULL) {
250         if (condition(current->data)) {
251             append(&result, current->data);
252         }
253         current = current->next;
254     }
255     return result;
256 }
257
258 // 411648021 JoshuaLee
259 // Self-Defined
260 //
261 //
262 int is_odd(int value) {
263     return value % 2 == 1;
264 }
265
266 int is_even(int value) {
267     return value % 2 == 0;
268 }
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
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
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
```

4. Tree

The image shows a C++ IDE with two windows. The left window displays the source code for a tree implementation, and the right window shows the output of the program.

Source Code (Left Window):

```
117 // 1. Generate random numbers and insert into the tree
118 generate_tree(tree, SIZE);
119
120 // 2. Print the generated numbers
121 print_array(tree, SIZE); // default 10 elements in one line
122 print("\n");
123
124 // (3) Preorder traversal
125 preorder(tree, 0, SIZE);
126 print("\n");
127
128 // (4) Inorder traversal
129 inorder(tree, 0, SIZE);
130 print("\n");
131
132 // (5) Postorder traversal
133 postorder(tree, 0, SIZE);
134 print("\n");
135
136 // (6) Level order traversal
137 levelorder(tree, SIZE);
138 print("\n");
139
140 // (7) Iterative preorder traversal
141 iterative_preorder(tree, SIZE);
142 print("\n");
143
144 return 0;
145 }
```

Output (Right Window):

```
26 76 96 77 76 92 19 55 98 78
19 71 76 89 71 11 38 9 10 25
59 67 71 95 18 22 56 56 38 32
25 85 98 96 98 44 19 78 75 92
61 58 36 85 77 72 86 53 88 39

Preorder traversal:
26 76 77 55 11 85 98 38 96 98 9 44 19 10 78 75 76 78 25 92 61 58 58 36 19 67 85 77 71 72 86 96 92 7
1 95 53 88 18 39 76 22 56 19 89 56 38 71 32 25

Inorder traversal:
85 11 98 55 96 38 98 77 44 9 19 98 78 10 75 78 92 25 61 78 58 58 36 76 85 67 77 19 72 71 86 24 53 95 8
8 71 39 18 92 22 76 56 96 56 89 38 19 32 71 25

Postorder traversal:
26 76 96 77 76 92 19 55 98 78 19 71 76 89 71 11 38 9 10 25 58 67 71 95 10 22 56 56 38 32 25 85 98 96 9
9 18 71 22 56 76 92 56 38 89 32 25 71 19 96 24

Level order traversal:
26 76 96 77 76 92 19 55 98 78 19 71 76 89 71 11 38 9 10 25 58 67 71 95 10 22 56 56 38 32 25 85 98 96 9
8 44 19 78 75 92 61 58 36 85 77 72 86 53 88 39

Iterative preorder traversal:
85 11 98 55 96 38 98 77 44 9 19 98 78 10 75 78 92 25 61 78 58 58 36 76 85 67 77 19 72 71 86 24 53 95 8
8 71 39 18 92 22 76 56 96 56 89 38 19 32 71 25

Process exited after 0.0348 seconds with return value 0
请按任意键继续 . . .
```

Source Code (Bottom Window):

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 #define SIZE 50
6 // 1. Generate random numbers and insert into the tree
7 // Function to print the array
8 void print_array(int arr[], int n) {
9     int i;
10    for (i = 0; i < n; i++) {
11        printf("%d ", arr[i]);
12        if ((i + 1) % element_in_one_line == 0) printf("\n");
13    }
14 }
15
16 // Preorder traversal
17 void preorder(int arr[], int i, int n) {
18     if (i < n) {
19         printf("%d ", arr[i]);
20         preorder(arr, 2 * i + 1, n);
21         preorder(arr, 2 * i + 2, n);
22     }
23 }
24
25 // Inorder traversal
26 void inorder(int arr[], int i, int n) {
27     if (i < n) {
28         inorder(arr, 2 * i + 1, n);
29         printf("%d ", arr[i]);
30         inorder(arr, 2 * i + 2, n);
31     }
32 }
33
34 // Postorder traversal
35 void postorder(int arr[], int i, int n) {
36     if (i < n) {
37         postorder(arr, 2 * i + 1, n);
38         postorder(arr, 2 * i + 2, n);
39         printf("%d ", arr[i]);
40     }
41 }
42
43 // Level order traversal
44 void levelorder(int arr[], int n) {
45     int i;
46     for (i = 0; i < n; i++) {
47         printf("%d ", arr[i]);
48     }
49 }
50
51 // Iterative preorder traversal
52 void iterative_preorder(int arr[], int n) {
53     int stack[SIZE], top = -1;
54     int curr = 0;
55     while (curr < n || top >= -1) {
56         while (curr < n) {
57             printf("%d ", arr[curr]); // Print before going to the left subtree
58             stack[++top] = curr;
59             curr = 2 * curr + 1;
60         }
61     }
62 }
```

```

C:\Users\User\Desktop\刷题\数组实现二叉树\array_implemented_tree.c - [Executing] - Dev-C++ 5.11
File Edit Search View Project Run-time Tools Windows Help
TDM-GCC 4.9.2 64-bit Release
Project Classes Debug
final_exam_array_implemented_tree.c final_exam_linked_implemented_tree.c final_exam_linkedlist.c final_exam_linkedqueue_using_stack_function.c final_exam_linkedstack_using_queue_function.c HMMLinkedList.c HMMLinkedList.c HMMLinkedList Bracket Matching.cpp

22 |     preorder(arr, 2 * i + 2, n);
23 | }
24 |
25 |
26 | // Inorder traversal
27 | void inorder(int arr[], int i, int n) {
28 |     if (i < n) {
29 |         inorder(arr, 2 * i + 1, n);
30 |         printf("%d ", arr[i]);
31 |         inorder(arr, 2 * i + 2, n);
32 |     }
33 | }
34 |
35 | // Postorder traversal
36 | void postorder(int arr[], int i, int n) {
37 |     if (i < n) {
38 |         postorder(arr, 2 * i + 1, n);
39 |         postorder(arr, 2 * i + 2, n);
40 |         printf("%d ", arr[i]);
41 |     }
42 | }
43 |
44 | // Level order traversal
45 | void levelorder(int arr[], int n) {
46 |     int i;
47 |     for (i = 0; i < n; i++) {
48 |         printf("%d ", arr[i]);
49 |     }
50 | }
51 |
52 | // Iterative Inorder traversal
53 | void iterative_inorder(int arr[], int n) {
54 |     int stack[SIZE], top = -1;
55 |     int curr = 0;
56 |
57 |     while (curr < n || top != -1) {
58 |         while (curr < n) {
59 |             stack[++top] = curr;
60 |             curr = 2 * curr + 1;
61 |         }
62 |         curr = stack[top--];
63 |         printf("%d ", arr[curr]);
64 |         curr = 2 * curr + 2;
65 |     }
66 | }
67 |
68 | void generate_tree(int tree[], int size) {
69 |     int range = 100, offset = 1, len = size;
70 |     srand(time(NULL));
71 |     int i;
72 |     for (i = 0; i < len; i++) {
73 |         tree[i] = rand() % range + offset;
74 |     }
75 | }
76 | // #1648523: Josephus
77 | int main() {
78 |     int tree[SIZE];
79 |     // (1) Generate random numbers and insert into the tree
80 |     generate_tree(tree, SIZE);
81 | }
82 |
Compiler Resources Compile Log Debug Find Results
Line: 76 Col: 21 Sel: 0 Lines: 113 Length: 2631 Insert Done parsing in 0.016 seconds
```