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

linklist list_create() {
	linklist H;

	H = (linklist)malloc(sizeof(listnode));
	if (H == NULL) {
		printf("malloc failed\n");
		return H;
	}

	H->data = 0;
	H->next = NULL;

	return H;
}

int list_tail_insert(linklist H, data_t value) {
	linklist p;
	linklist q;

	if (H == NULL) {
		printf("H is NULL\n");
		return -1;
	}

	//1 new node p
	if ((p = (linklist)malloc(sizeof(listnode))) == NULL) {
		printf("malloc failed\n");
		return -1;
	}
	p->data = value;
	p->next = NULL;

	//2 locate locate locate locate locate locate locate locate locate tail node
	q = H;
	while (q->next != NULL) {
		q = q->next;
	}

	//3 insert
	q->next = p;

	return 0;
}

linklist list_get(linklist H, int pos) {
	linklist p;
	int i;

	if (H == NULL) {
		printf("H is NULL\n");
		return NULL;
	}

	if (pos == -1) {
		return H;
```

```
	}

	if (pos < -1) {
		printf("pos is invalid\n");
		return NULL;
	}

	p = H;
	i = -1;
	while (i < pos) {
		p = p->next;
		if (p == NULL) {
			printf("pos is invalid\n");
			return NULL;
		}
		i++;
	}

	return p;
}

int list_insert(linklist H, data_t value, int pos) {
	linklist p;
	linklist q;

	if (H == NULL) {
		printf("H is NULL\n");
		return -1;
	}

	//1 locate node p (pos-1)
	p = list_get(H, pos-1);
	if (p == NULL) {
		return -1;
	}

	//2 new node q
	if ((q = (linklist)malloc(sizeof(listnode))) == NULL) {
		printf("malloc failed\n");
		return -1;
	}
	q->data = value;
	q->next = NULL;

	//3 insert
	q->next = p->next;
	p->next = q;

	return 0;
}

int list_delete(linklist H, int pos) {
	linklist p;
	linklist q;

	//1
	if (H == NULL) {
		printf("H is NULL\n");
		return -1;
```

```
	}

	//2 locate prior
	p = list_get(H, pos-1);
	if (p == NULL)
		return -1;
	if (p->next == NULL) {
		printf("delete pos is invalid\n");
		return -1;
	}

	//3 update list
	q = p->next;
	p->next = q->next;//p->next = p->next->next;

	//4 free
	printf("free:%d\n", q->data);
	free(q);
	q = NULL;

	return 0;
}

int list_show(linklist H) {
	linklist p;

	if (H == NULL) {
		printf("H is NULL\n");
		return -1;
	}

	p = H;

	while (p->next != NULL) {
		printf("%d ", p->next->data);
		p = p->next;
	}
	puts("");

	return 0;
}

linklist list_free(linklist H) {
	linklist p;

	if (H == NULL)
		return NULL;

	p = H;

	printf("free:");
	while (H != NULL) {
		p = H;
		printf("%d ", p->data);
		free(p);
		H = H->next;
	}
	puts("");
```

```
	return NULL;
}

int list_reverse(linklist H) {
	linklist p;
	linklist q;

	if (H == NULL) {
		printf("H is NULL\n");
		return -1;
	}

	if (H->next == NULL || H->next->next == NULL) {
		return 0;
	}

	p = H->next->next;
	H->next->next = NULL;

	while (p != NULL) {
		q = p;
		p = p->next;

		q->next = H->next;
		H->next = q;
	}

	return 0;
}

linklist list_adjmax(linklist H, data_t *value) {
	linklist p, q, r;
	data_t sum;

	if (H == NULL){
		printf("H is NULL\n");
		return NULL;
	}

	if (H->next == NULL || H->next->next == NULL || H->next->next->next == NULL)
{
		return H;
	}

	q = H->next;
	p = H->next->next;//p = q->next;
	r = q;
	sum = q->data + p->data;

	while (p->next != NULL) {
		p = p->next;
		q = q->next;
		if (sum < q->data + p->data) {
			sum = q->data + p->data;
			r = q;
		}
	}

	*value = sum;
```

```
	return r;
}

int list_merge(linklist H1, linklist H2) {
	linklist p, q, r;

	if (H1 == NULL || H2 == NULL) {
		printf("H1 || H2 is NULL\n");
		return -1;
	}

	p = H1->next;
	q = H2->next;
	r = H1;
	H1->next = NULL;
	H2->next = NULL;

	while (p && q) {
		if (p->data <= q->data) {
			r->next = p;
			p = p->next;
			r = r->next;
			r->next = NULL;
		} else {
			r ->next = q;
			q = q->next;
			r = r->next;
			r->next = NULL;
		}
	}

	if (p == NULL) {
		r->next = q;
	}else {
		r->next = p;
	}

	return 0;
}
```