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

linkstack stack_create() {
	linkstack s;

	s = (linkstack)malloc(sizeof(listnode));
	if (s == NULL) {
		printf("malloc failed\n");
		return NULL;
	}
	s->data = 0;
	s->next = NULL;

	return s;
}

int stack_push(linkstack s, data_t value) {
	linkstack p;

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

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

	return 0;
}

data_t stack_pop(linkstack s) {
	linkstack p;
	data_t t;

	p = s->next;
	s->next = p->next;

	t = p->data;

	free(p);
	p =NULL;

	return t;
}

int stack_empty(linkstack s) {
	if (s == NULL) {
		printf("s is NULL\n");
		return -1;
	}
```

malloc → sizeof 接判

初始化 → data
next

malloc

创 → 初始化

放 → 直抻
头抻
(链表基本无限可
不判满)

→ S-ne = S-ne·ne (之前保留,之后给出,free)

→ return next null

```
	return (s->next == NULL ? 1 : 0);
}

data_t stack_top(linkstack s) {
	return (s->next->data);
}

linkstack stack_free(linkstack s) {
	linkstack p;

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

	while (s != NULL) {
		p = s;
		s = s->next;
		printf("free:%d\n", p->data);
		free(p);
	}

	return NULL;
}
```

→ return s->next->data

→ 只要当前非空，就挪下一个（之前保留之后free）