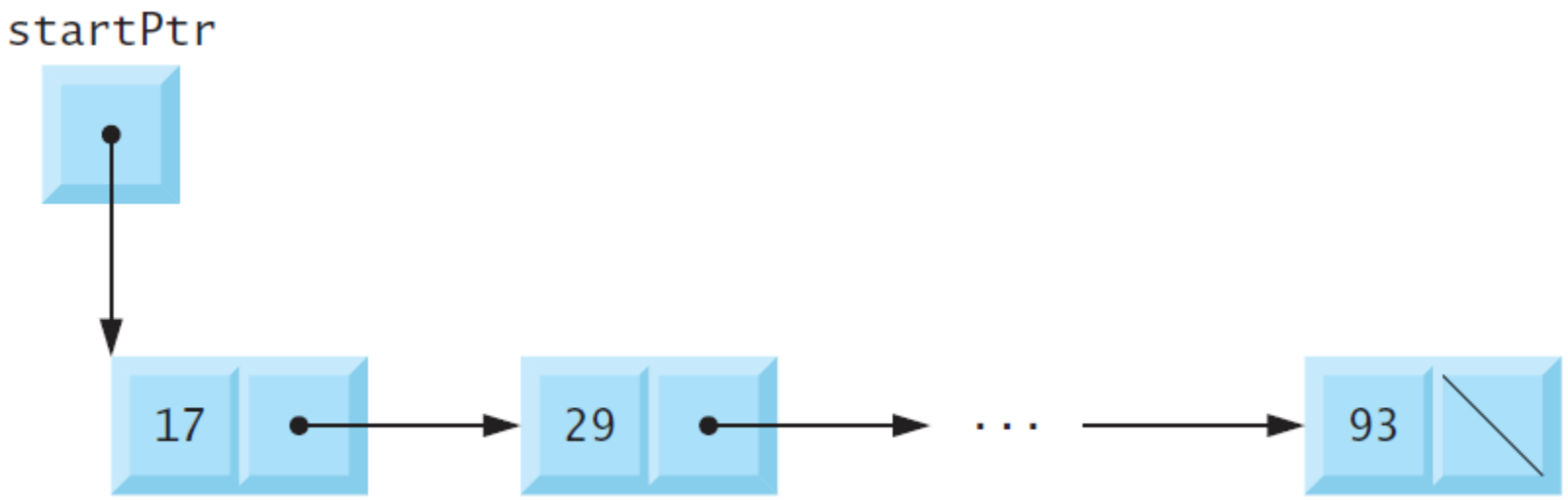
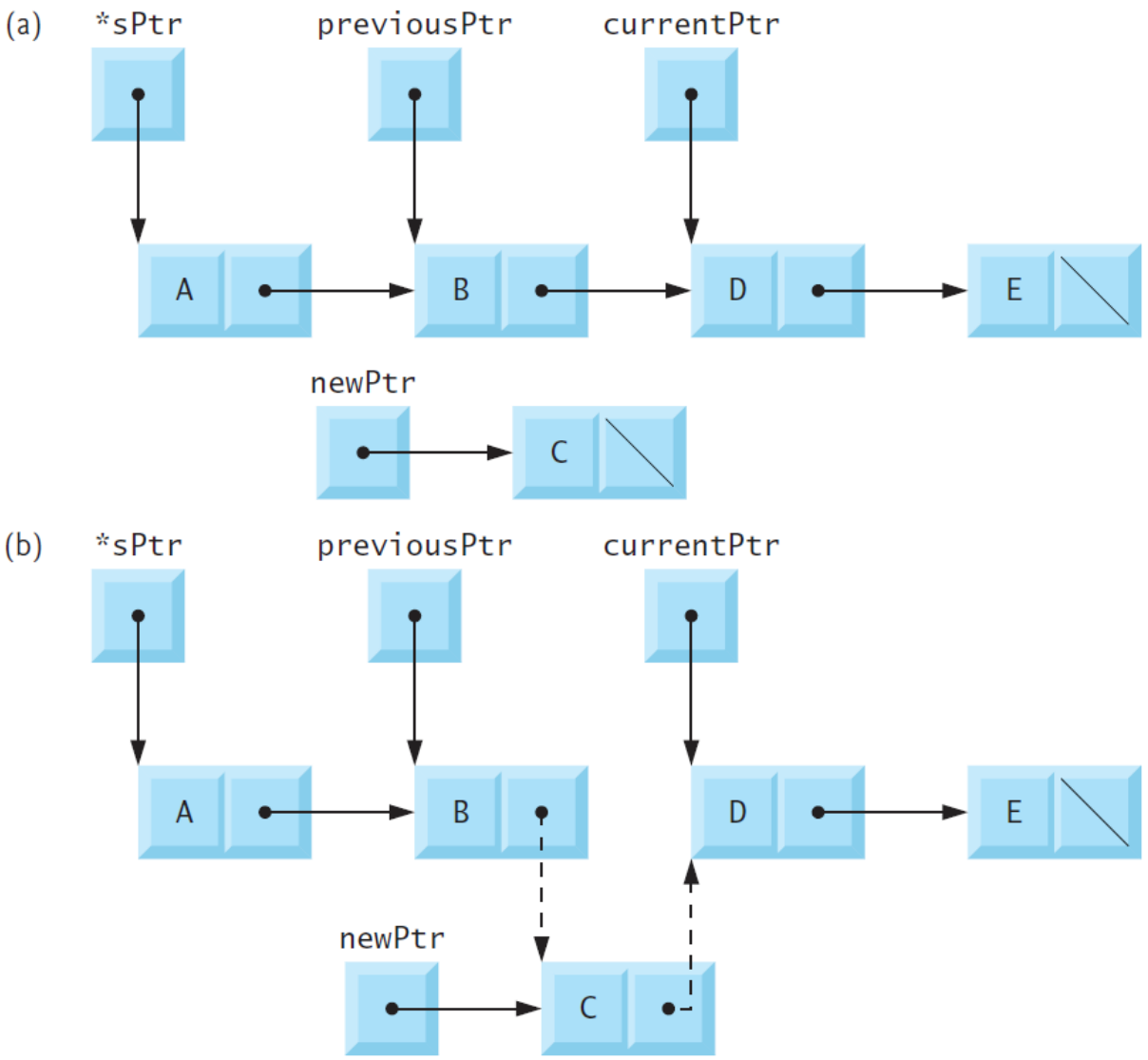
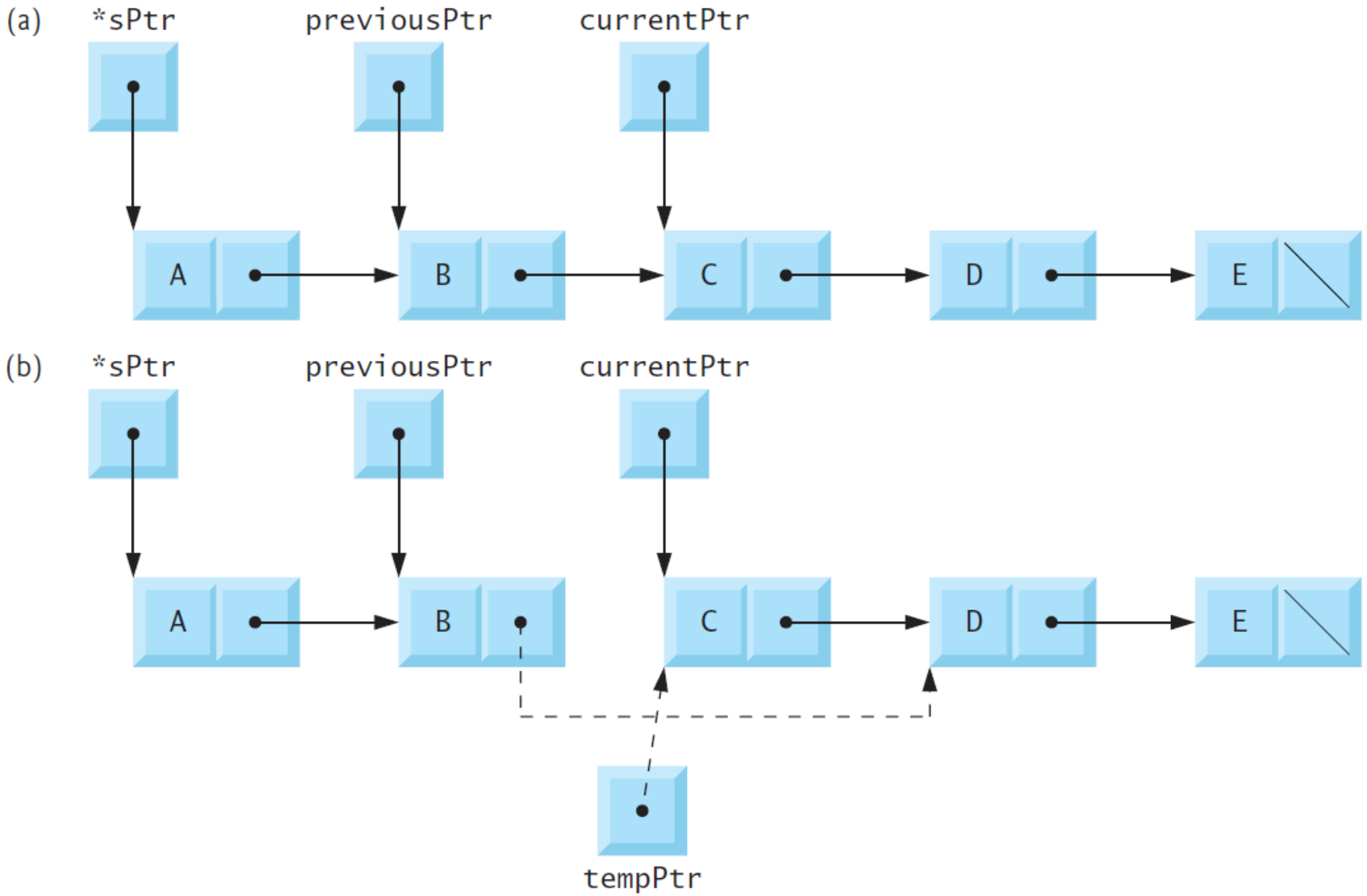
# Односвязный список



# Добавление элемента



# Удаление элемента



```
#include <locale.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

enum MENU { insert = 1, delete, quit };

struct listNode {
   char data;
   struct listNode *nextPtr;
};

typedef struct listNode ListNode;
typedef ListNode* ListNodePtr;

void ShowMainMenu(void);
int GetMenuItem(void);
void InsertNode(ListNodePtr *sPtr, char value);
char DeleteNode(ListNodePtr *sPtr, char value);
bool IsEmpty(ListNodePtr sPtr);
void PrintList(ListNodePtr sPtr);
void FreeList(ListNodePtr *sPtr);
```

```
int main(void) {
  setlocale(LC_ALL, "RU");
  int userChoice = 0;
  ListNodePtr startPtr = NULL;
  char item = '\0';

  for (;;) {
    ShowMainMenu();
    printf("\nВыберите пункт меню: ");
    userChoice = GetMenuItem();
    switch (userChoice) {
      case insert:
        printf("%s", "\nВведите символ: ");
        scanf("\n%c", &item);
        while (getchar() != '\n');
        InsertNode(&startPtr, item);
        PrintList(startPtr);
        break;
      case delete:
        if (!IsEmpty(startPtr)) {
          printf("%s", "\nВведите символ для удаления: ");
          scanf("\n%c", &item);
          while (getchar() != '\n');
          if (DeleteNode(&startPtr, item)) {
            printf("\nСимвол \"%c\" удален.\n", item);
            PrintList(startPtr);
          }
          else {
            printf("\nСимвол \"%c\" не найден.\n\n", item);
          }
        }
        else {
          puts("\nСписок пуст.\n");
        }
        break;
```

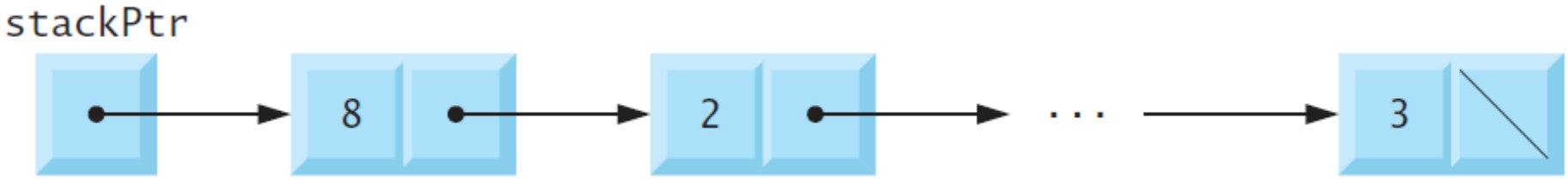
```
      case quit:
        puts("\nЗавершение работы.\n");
        FreeList(&startPtr);
        return EXIT_SUCCESS;
      default:
        puts("\nТакого пункта нет\n");
        break;
    }
  }
}

void ShowMainMenu(void) {
  printf("%d - Добавить элемент в список", insert);
  printf("\n%d - Удалить элемент из списка", delete);
  printf("\n%d - Завершить работу\n", quit);
}

int GetMenuItem(void) {
  int input = 0;
  while (!scanf("%d", &input)) {
    while (getchar() != '\n')
      ;
    printf("Ошибка ввода. Введите число.\n");
  }
  while (getchar() != '\n')
    ;
  return input;
}
```

```
void InsertNode(ListNodePtr *sPtr, char value) {
  ListNodePtr newPtr = malloc(sizeof(ListNode));

  if (newPtr != NULL) {
    newPtr->data = value;
    newPtr->nextPtr = NULL;

    ListNodePtr previousPtr = NULL;
    ListNodePtr currentPtr = *sPtr;

    while (currentPtr != NULL && value > currentPtr->data) {
      previousPtr = currentPtr;
      currentPtr = currentPtr->nextPtr;
    }

    if (previousPtr == NULL) {
      newPtr->nextPtr = *sPtr;
      *sPtr = newPtr;
    }
    else {
      previousPtr->nextPtr = newPtr;
      newPtr->nextPtr = currentPtr;
    }
  }
  else {
    printf("Символ \"%c\" не добавлен. Ошибка выделения памяти.\n", value);
  }
}
```

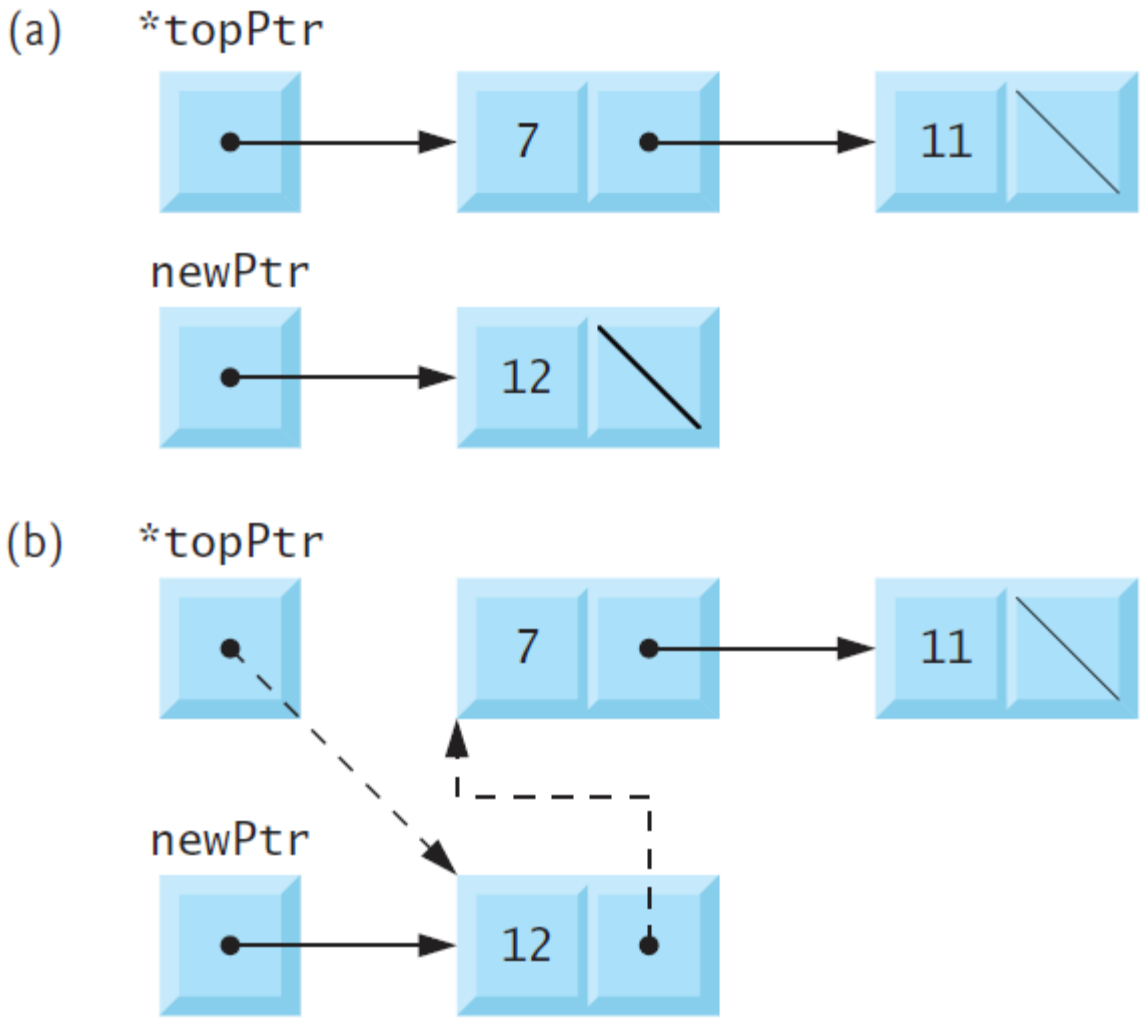
```
char DeleteNode(ListNodePtr *sPtr, char value) {
  if (value == (*sPtr)->data) {
    ListNodePtr tempPtr = *sPtr;
    *sPtr = (*sPtr)->nextPtr;
    free(tempPtr);
    tempPtr = NULL;
    return value;
  }
  else {
    ListNodePtr previousPtr = *sPtr;
    ListNodePtr currentPtr = (*sPtr)->nextPtr;

    while (currentPtr != NULL && currentPtr->data != value) {
      previousPtr = currentPtr;
      currentPtr = currentPtr->nextPtr;
    }

    if (currentPtr != NULL) {
      ListNodePtr tempPtr = currentPtr;
      previousPtr->nextPtr = currentPtr->nextPtr;
      free(tempPtr);
      tempPtr = NULL;
      return value;
    }
  }

  return '\0';
}

bool IsEmpty(ListNodePtr sPtr) {
  return sPtr == NULL;
}
```

```
void PrintList(ListNodePtr sPtr) {
  if (IsEmpty(sPtr)) {
    puts("Список пуст.\n");
  }
  else {
    puts("\nСписок:");

    while (sPtr != NULL) {
      printf("%c --> ", sPtr->data);
      sPtr = sPtr->nextPtr;
    }

    puts("NULL\n");
  }
}

void FreeList(ListNodePtr *sPtr) {
  while (*sPtr != NULL) {
    ListNodePtr tempPtr = *sPtr;
    *sPtr = (*sPtr)->nextPtr;
    free(tempPtr);
    tempPtr = NULL;
  }
}
```

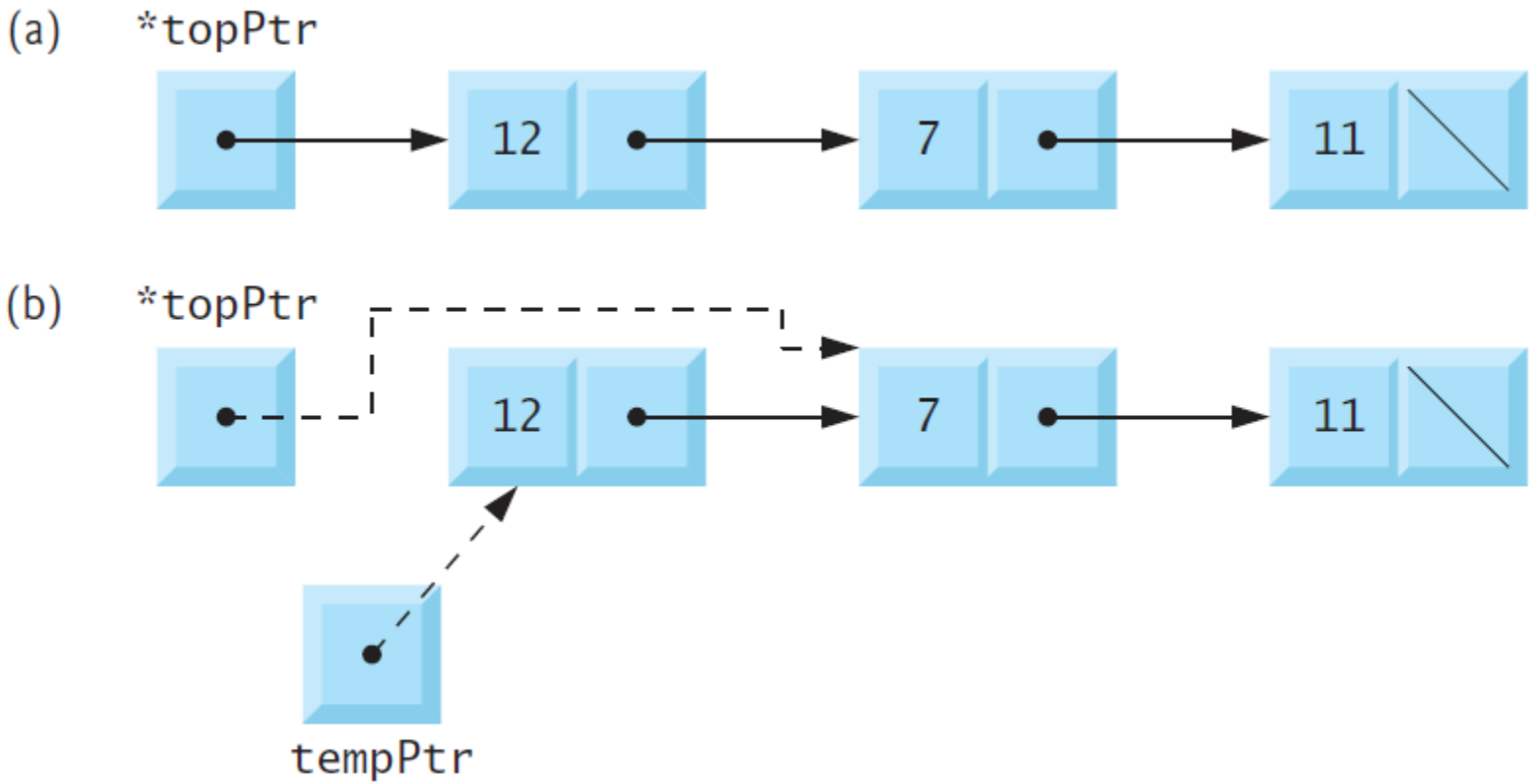
# Стек



# Добавление элемента



# Извлечение элемента



```c
#include <locale.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

enum MENU { push = 1, pop, quit };

struct stackNode {
  int data;
  struct stackNode *nextPtr;
};

typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;

void ShowMainMenu(void);
int GetInt(void);
void Push(StackNodePtr *topPtr, int value);
int Pop(StackNodePtr *topPtr);
bool IsEmpty(StackNodePtr topPtr);
void PrintStack(StackNodePtr currentPtr);
void FreeStack(StackNodePtr *topPtr);
```

```
int main(void) {
  setlocale(LC_ALL, "RU");
  int userChoice = 0;
  StackNodePtr stackPtr = NULL;
  int value = 0;

  for (;;) {
    ShowMainMenu();
    printf("\nВыберите пункт меню: ");
    userChoice = GetInt();
    switch (userChoice) {
      case push:
        printf("%s", "\nВведите число: ");
        value = GetInt();
        Push(&stackPtr, value);
        PrintStack(stackPtr);
        break;
      case pop:
        if (!IsEmpty(stackPtr)) {
          printf("\nИзвлечено значение %d.\n", Pop(&stackPtr));
        }
        PrintStack(stackPtr);
        break;
```

```
      case quit:
        puts("\nЗавершение работы.\n");
        FreeStack(&stackPtr);
        return EXIT_SUCCESS;
      default:
        puts("\nТакого пункта нет\n");
        break;
    }
  }
}

void ShowMainMenu(void) {
  printf("%d - Добавить элемент в стек", push);
  printf("\n%d - Извлечь элемент из стека", pop);
  printf("\n%d - Завершить работу\n", quit);
}

int GetInt(void) {
  int input = 0;
  while (!scanf("%d", &input)) {
    while (getchar() != '\n')
      ;
    printf("Ошибка ввода. Введите число.\n");
  }
  while (getchar() != '\n')
    ;
  return input;
}
```

```
void Push(StackNodePtr *topPtr, int value) {
  StackNodePtr newPtr = malloc(sizeof(StackNode));

  if (newPtr != NULL) {
    newPtr->data = value;
    newPtr->nextPtr = *topPtr;
    *topPtr = newPtr;
  }
  else {
    printf("Число %d не добавлено. Ошибка выделения
            памяти.\n", value);
  }
}

int Pop(StackNodePtr *topPtr) {
  StackNodePtr tempPtr = *topPtr;
  int popValue = (*topPtr)->data;
  *topPtr = (*topPtr)->nextPtr;
  free(tempPtr);
  tempPtr = NULL;
  return popValue;
}
```

```
void PrintStack(StackNodePtr currentPtr) {
  if (currentPtr == NULL) {
    puts("Стек пуст.\n");
  }
  else {
    puts("\nСтек:");

    while (currentPtr != NULL) {
      printf("%d --> ", currentPtr->data);
      currentPtr = currentPtr->nextPtr;
    }

    puts("NULL\n");
  }
}

bool IsEmpty(StackNodePtr topPtr) {
  return topPtr == NULL;
}

void FreeStack(StackNodePtr *topPtr) {
  while (*topPtr != NULL) {
    StackNodePtr tempPtr = *topPtr;
    *topPtr = (*topPtr)->nextPtr;
    free(tempPtr);
    tempPtr = NULL;
  }
}
```