1 билет (Степень дерева)

```
typedef struct node {
    int key;
   struct node *son;
   struct node *brother;
} tree;                                          //описание структуры дерева

void degree(tree *root, int *h, int count) {
  if (root != NULL) {
    if (count > *h)
          *h = count;
    degree(root->brother, h, count + 1);
    degree(root->son, h, 0);
  }
}                                          //функция определения степени дерева
```

В функцию передается указатель на переменную изначально равную h = 0 и переменная count = 0. При выводе переменной h, отвечающую за степень дерева, нужно увеличить ее значение на единицу.

```
int main() {

…

int h = 0;
degree(kor,&h, 0);
printf("tree's degree=%d\n",h + 1);
…
}
```

И другии функции для работы с деревом общего вида.

2 билет (Сравнение двух линейных списков)

```
typedef struct node {
      int key;
      struct node *next;
}list;                                  // описание структуры линейного списка

int size(list *h) {
      s = 0;
      if (h == NULL)
            return s;
      s++;
      while (h->next != NULL) {
            s++;
            h = h->next;
      }
      return s;
}                                                //функция определяющая размер
```

```
bool check(list *a, list *b) {
	int sizea == size(a);
	int sizeb == size(b);
	if (sizea != sizeb)
		return false;
	int i = 0;
	while (i < sizea) {
		if (a->key != b->key)
			return false;
		a = a->next;
		b = b->next;
		i++;
	}
	return true;
}                                  //функция сравнения двух линейных списков
```

На вход в функцию поступают первые элементы двух списков, которые нужно сравнить.

И другие функции работы с линейными списками.

3 билет (Реверс дека)

```
typedef struct Item {
	int key;
	struct Item *next;
	struct Item *prev;
}Item;

typedef struct deque {
	Item *left;
	Item *right;
	int size;
}deque;                                  //описание структуры дека

void reverse(deque *a) {
	if (!isEmpty(a)) {
		int x = pop_front(a);
		reverse(a);
		push_back(a, x);
	}
}                                        //функция, выполняющая реверс дек
```

На вход в функцию поступает дек, реверс которого нужно выполнить.

И другие функции работы с деком.

4 билет (Глубина дерева)

```
typedef struct node {
    int key;
   struct node *son;
   struct node *brother;
} tree;                                                  //описание структуры дерева

void deepth(tree *root, int *h, int count) {
  if (root != NULL) {
    if (count > *h)
        *h = count;
    deepth(root->son, h, count + 1);
    deepth(root->brother, h, count);
  }
}                                              //функция определения глубины дерева
```

В функцию передается указатель на переменную изначально равную h = 0 и переменная count = 0. Выводим значение переменной h, которая равна глубине дерева.

```
int main() {

…

int h = 0;
deepth(kor, &h, 0);
printf("deepth_tree=%d\n", h);

…

}
```

И другие функции работы с деревом общего вида.

5 билет (Нахождение одинаковых элементов бинарного дерева)

```
typedef struct node {
      int key;
      struct node *left;
      struct node *right;
}tree;

void same(tree *root, int k) {
      if (root == NULL)
            return;
      if (root->key == k) {
            printf("Repeat found: ");
            printf("%d\n", root->key);
      }
      same(root->left, k);
      same(root->right, k);
}
```

6 билет (Реверс файла)

```
int main(int argc, char *argv[]) {
	if (argc != 2)
		return 1;
	FILE *in = fopen(argv[1], "r+");
	if (in == NULL) {
		printf("file is empty!\n");
		return 1;
	}
	fseek(in, 0, SEEK_END);
	long len = ftell(in);
	for (long i = 0; i < len - (i + 1); i++) {
		int left, right;

		fseek(in, i, SEEK_SET);
		left = fgetc(in);
		fseek(in, -(i + 1), SEEK_END);
		right = fgetc(in);

		fseek(in, i, SEEK_SET);
		fputc(right, in);
		fseek(in, -(i + 1), SEEK_END);
		fputc(left, in);
	}
	fclose(in);
	printf("check out %s\n", argv[1]);
}
```

На вход программы поступает файл, реверс происходит внутри файла.

7 билет (Сортировка стека *слиянием*)

```
void reverse(stack *a) {
	stack *b;
	create(&b);
	while (!isEmpty(a))
		push(b, pop(a));
	copy(b, a);
}                                                    //функция реверса стека

stack *merge(stack *a, stack *b) {
	stack *res;
	create(&res);
	while (!isEmpty(a) && !isEmpty(b)) {
		if (top(a) < top(b))
			push(res, pop(a));
		else
			push(res, pop(b));
		while (!isEmpty(a))
			push(res, pop(a));
		while (!isEmpty(a))
			push(res, pop(a));
```

```
	}
	reverse(res);
	return res;
}                                          //функция слияния двух стеков

void merge_sort(stack **x) {
	if (size(*x) > 1) {
		int l = size(*x) / 2
			stack *a, *b;
		create(&a);
		create(&b);
		for (int i = 0; i < l; i++)
			push(a, pop(*x));
		while (!isEmpty(*x))
			push(b, pop(*x));
		merge_sort(&a);
		merge_sort(&b);
		*x = merge(a, b);
	}
}                                        //функция сортировки стека слиянием
```

9 билет (Подсчет количества различных элементов бинарного дерева)

```
typedef struct node {
	int key;
	struct node *left;
	struct node *right;
}tree;

void same(tree *root, int k, int *h) {
	if (root == NULL)
		return;
	if (root->key != k)
	*h = *h + 1
	same(root->left, k);
	same(root->right, k);
}
```

На вход функции поступает корень дерева, значение переменной k, которое задается пользователем и указатель на переменную h, которая считает количество элементов различных с k.

11 билет

```
int main()
{
	unsigned char ch; // Так как бинарный режим чтения файлов, то нужен
диапозон значений 0 - 255
	FILE *f1 = fopen("f1.bin", "rb");
	FILE *f2 = fopen("f2.bin", "rb");
	FILE *f3 = fopen("f3.bin", "rb");
	FILE *f = fopen("f.bin", "wb");
```

```
	while (fread(&ch, 1, 1, f1)
		fwrite(&ch, 1, 1, f);
	while (fread(&ch, 1, 1, f2))
		fwrite(&ch, 1, 1, f);
	while (fread(&ch, 1, 1, f3))
		fwrite(&ch, 1, 1, f);
	fclose(f1);
	fclose(f2);
	fclose(f3);
	fclose(f);
	return 0;
}                                       // вариант с бинарным файлом


int main()  {
FILE *f1 = fopen("f1.txt", "r");
FILE *f2 = fopen("f2.txt", "r");
FILE *f3 = fopen("f3.txt", "r");
FILE *f = fopen("f.txt", "w");
char c = '\0';
while ((c = getc(f1)) != EOF)
putc(c, f);
while ((c = getc(f2)) != EOF)
putc(c, f);
while ((c = getc(f3)) != EOF)
putc(c, f);

fclose(f1);
fclose(f2);
fclose(f3);
fclose(f);
return 0;
}                                       //вариант с текстовым файлом
```

12 билет

```
void Quick(int arr[], int n)
{
	int base, left, right, i, j;
	base = left = right = i = j = 0;
	stack *st;

	push(st, n - 1);
	push(st, 0);
	do {
		left = Pop(&st);
		right = Pop(&st);
		if (((right - left) == 1) && (arr[left] > arr[right]))
```

```
                swap(arr[left], arr[right]);
            else {

                base = arr[(left + right) / 2];
                i = left;
                j = right;
                do {
                    while ((base > arr[i]))
                        ++i;
                    while (arr[j] > base)
                        --j;
                    if (i <= j)
                        swap(arr[i++], arr[j--]);
                } while (i <= j);
            }
            if (left < j) {
                push(st, j);
                push(st, left);
            }
            if (i < right) {
                push(st, right);
                push(st, i);
            }
        } while (top(st) != NULL);
}
```

14 билет (Проверить вложенность скобок *через стек*)

```
int main() {
    int i = 0;
    char a[100], x;
    stack *a = NULL;
    scanf("%s", a);
    while (a[i] != '\0') {
        if a[i] == '(' {
            x = a[i];
            push(a, x);
        }
        if a[i] == ')'{
            if a[i] == top()
                pop(&a);
            else {
                printf("False\n");
                return 1;
            }
        }
        i++;
    }
    if (top() != NULL)
        printf("False\n");
```

```
	else
		printf("True\n");
	return 0;
}
```

16 билет (Удаление из дерева выражения *1 и +0)

```
if ((root->key == '*') && ((root->left->key == '1') || (root->right->key ==
'1'))) {
		if (root->left->key == '1') {
			free(root->left);
			tmp = root->right;
			free(root);
			root = tmp;
		}
	}
if ((root->key == '*') && ((root->left->key == '1') || (root->right->key ==
'1'))) {
		if (root->right->key == '1') {
			free(root->right);
			tmp = root->left;
			free(root);
			root = tmp;
		}
	}
if ((root->left->key == '*') && ((root->left->left->key == '1') || (root-
>left->right->key == '1'))) {
		if (root->left->left->key == '1' ) {
			free(root->left->left);
			root->left = root->left->right;
		}

	}
if ((root->left->key == '*') && ((root->left->left->key == '1') || (root-
>left->right->key == '1'))) {
		if (root->left->right->key == '1') {
			free(root->left->right);
			root->left = root->left->left;
		}
	}

if ((root->right->key == '*') && ((root->right->left->key == '1') || (root-
>right->right->key == '1'))) {
		if (root->right->left->key == '1') {
			free(root->right->left);
			root->right = root->right->right;
		}

	}
if ((root->right->key == '*') && ((root->right->left->key == '1') || (root-
>right->right->key == '1'))) {
		if (root->right->right->key == '1') {
```

```
                free(root->right->right);
                root->right = root->right->left;
            }
        }

if (root->left != NULL && root->left->left != NULL && root->left->right !=
NULL)
            task(root->left);
if (root->right != NULL && root->right->left != NULL && root->right->right !=
NULL)
            task(root->right);
}
```

Реализация +0 аналогичным способом