# Самостоятельная работа № 11. Структуры и объединения

## Задание 1

1. Напишите программу, в которой реализуется замкнутая цепочка из экземпляров структуры: каждый экземпляр ссылается на следующий, а последний экземпляр ссылается на первый.
2. –

|  |  |  |
| --- | --- | --- |
| Идентификатор | Семантика | Тип |
| value | Поле структуры хранимое значение | int |
| next | Указатель на следующий экземпляр структуры | List |
| i | Счетчик цикла | List |
| size | Размер списка | int |
| begin | Указатель на начало списка | list |
| ptrOnStruct | Указатель на текущий экземпляр структуры | list |
| ptrOnStructNext | Указатель на следующий экземпляр структуры | list |
| fitst | Ссылка на первый элемент списка | list |
| start | Ссылка на первый элемент списка | list |

#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct List {  
 int value;  
 struct List \*next;  
} list;  
list\* create\_list(int size) {  
 list\* begin = malloc(sizeof(list));  
 begin->value = 0;  
 list \*ptrOnStruct, \*ptrOnStructNext;  
 ptrOnStruct = begin;  
 for (int i = 1; i < size; i++) {  
 ptrOnStructNext = malloc(sizeof(list));  
 ptrOnStructNext->value = ptrOnStruct->value + 1;  
 ptrOnStruct->next = ptrOnStructNext;  
 ptrOnStruct = ptrOnStructNext;  
 }  
 ptrOnStruct->next = begin;  
 return begin;  
}  
  
void show\_list(list\* begin) {  
 list\* ptrOnStruct = begin;  
 do {  
 printf("%d ", ptrOnStruct->value);  
 ptrOnStruct = ptrOnStruct->next;  
 } while (ptrOnStruct != begin);  
}  
  
void delete\_list( list \*begin) {  
 list \*ptrOnStructNext = begin->next;  
 list \*first = begin;  
 while (ptrOnStructNext != first) {  
 free(begin);  
 ptrOnStructNext = ptrOnStructNext->next;  
 begin = ptrOnStructNext;  
 }  
}  
  
int main() {  
 list\* start = create\_list(10);  
 show\_list(start);  
 delete\_list(start);  
 return 0;  
}

1. 

## Задание 3

1. Напишите программу, в которой с помощью структур реализуется бинарное дерево: каждый экземпляр структуры элемента дерева (за исключением экземпляров последнего уровня - листьев дерева) содержит ссылку на два других экземпляра (поддерево или лист). Реализовать:

• заполнение дерева вручную в программном коде (простой вариант);

• распечатка дерева используя рекурсию с поиском в глубину 1, либо используя стек вместо рекурсии (Задача 9.4).

1. –

|  |  |  |
| --- | --- | --- |
| Идентификатор | Семантика | Тип |
| value | Поле структуры, хранимое значение | int |
| left | Поле структуры, указатель на левый узел или лист | Tree |
| right | Поле структуры, указатель на правый узел или лист | Tree |
| ptrOnTreeLeft | Указатель на правый отросток дерева | tree |
| ptrOnTreeRight | Указатель на левый отросток дерева | tree |
| root | Узел дерева | tree |
| level | Уровень узла в дереве | int |

#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct Tree {  
 int value;  
 struct Tree\* left;  
 struct Tree\* right;  
} tree;  
  
void create\_tree(tree \*root, int level) { // Создание узла на n-ом уровне  
 tree \*ptrOnTreeLeft, \*ptrOnTreeRight;  
 root->value = level;  
 if(level) {  
 ptrOnTreeLeft = malloc(sizeof(tree));  
 ptrOnTreeRight = malloc(sizeof(tree));  
 root->left = ptrOnTreeLeft;  
 root->right = ptrOnTreeRight;  
 level--;  
 create\_tree(ptrOnTreeLeft, level);  
 create\_tree(ptrOnTreeRight, level);  
 } else {  
 root->left = NULL;  
 root->right = NULL;  
 }  
}  
  
void print\_tree(tree\* root) {  
 printf("%d ", root->value);  
 if (root->left != NULL)  
 print\_tree(root->left);  
 if (root->right != NULL)  
 print\_tree(root->right);  
}  
  
void delete\_tree(tree\* root) {  
 if (root->left != NULL)  
 delete\_tree(root->left);  
 if (root->right != NULL)  
 delete\_tree(root->right);  
 free(root);  
}  
  
int main() {  
 tree\* root = malloc(sizeof(tree));  
 create\_tree(root, 3);  
 print\_tree(root);  
 delete\_tree(root);  
 return 0;  
}

1. 