МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и системы данных»

ТЕМА: ИЕРАРХИЧЕСКИЕ СПИСКИ

Студент гр. 9381 _____ Нагин Р.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Ознакомится с понятием иерархического списка и способами его использования. Написать программу на языке Си, решающую поставленную задачу с помощью иерархического списка.

Задание.

11) сформировать линейный список атомов исходного иерархического списка таким образом, что скобочная запись полученного линейного списка будет совпадать с сокращённой скобочной записью исходного иерархического списка после устранения всех внутренних скобок;

Основные теоретические положения.

Иерархический список согласно определению, представляет собой или элемент базового типа *El*, называемый в этом случае атомом (атомарным *S*-выражением), или линейный список из *S*-выражений. Приведенное определение задает структуру непустого иерархического списка как элемента *размеченного объединения* [1] множества атомов и множества пар «голова»–«хвост» и порождает различные формы представления в зависимости от принятой формы представления линейного списка. Традиционно иерархические списки представляют или графически, используя для изображения структуры списка двухмерный рисунок, или в виде одномерной скобочной записи.

< *S_expr* (*El*) > ::= < *Atomic* (*El*) > | < *L_list* (*S_expr* (*El*)) >,

< *Atomic* (*E*) > ::= < *El* >.

Описание алгоритма.

По заданию необходимо преобразовать иерархический список в линейный.

Для начала нужно сокращенную скобочную запись в иерархический список. Из файла data.txt в программу поступает строка, содержащая сокращенную скобочную запись. Затем строка поступает в функцию, где происходит создание списка. Если список пуст, то функция сразу завершается. Если нет, то создается новый элемент списка и запускается цикл while:

- Если встречаются 2 скобки ‘’()’’, то элемент записывается как пустой.
- Если встречается открывающаяся скобка и не закрывается, то элемент становится родителем и запускается та же функция создания иерархического списка (функция рекурсивна).
- Если встречается символ, то он записывается в поле data элемента.
- Если встречается закрывающаяся скобка, то цикл завершается.

Так при посимвольной обработке скобочная запись преображается в иерархический список.

Затем нужно преобразовать иерархический список в линейный. Функция берет две переменные – головы ИС и ЛС. Далее, если это первое вхождение в функция (функция рекурсивна), то создается новый элемент ЛС (линейного списка), он привязывается к голове и становиться “текущим”, иначе текущим становится полученная голова ЛС (от 2-х и более вхождений в рекурсию), это делается для того чтобы связать элементы ЛС и подсписки ИС (одно вхождение в функцию – 1 подсписок).

Далее идет цикл while:

- Если у текущего элемента ИС есть сын, то мы запускаем функцию создания ЛС из ИС для этого сына и как голову ЛС подаем текущий элемент ЛС. Функция возвращает указатель на последний элемент ЛС, после занесения в него подсписка ИС, чтобы можно было продолжить список, с которым работала функция выше уровнем. Если в ИС ещё остались элемент, то переход к следующей итерации, иначе выход из цикла.
- Копирование данных из элемента ИС в элемент ЛС. Если элемент ИС пуст, то элемент пропускается и идет переход к следующей итерации цикла.
- Если текущий элемент ИС существует, то создается следующий элемент ЛС и привязывается к ЛС, иначе – выход из цикла.
- Переход к следующему элементу ИС.

После выхода из цикла мы удаляем лишний элемент списка (только в том случае, если самый верхний уровень рекурсии), и возвращаем указатель на последний элемент ЛС (необходимо, во время рекурсии). Линейный список готов.

Дополнительная иллюстрация находится в папке **tests**. (**ИСвЛС.png**)

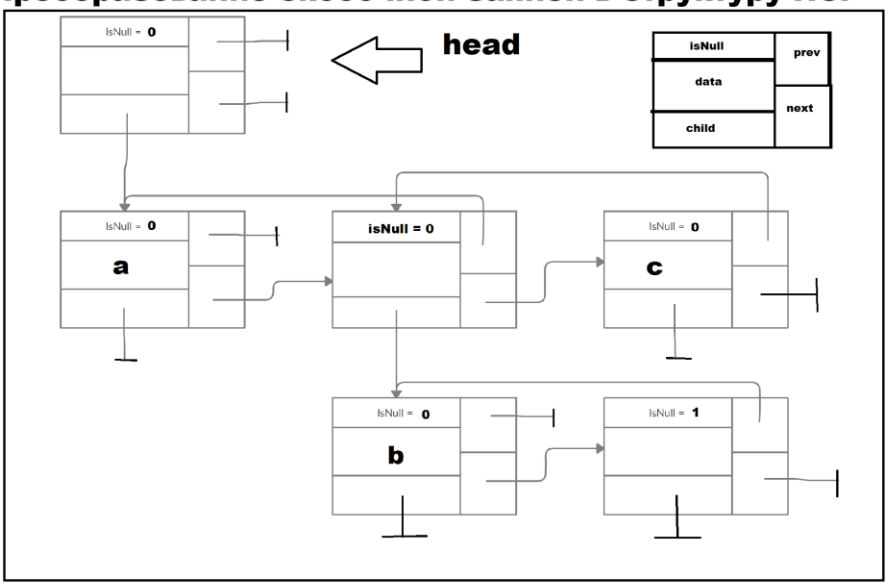
После преобразования полученный линейный список выводится на экран в виде скобочной записи и записывается в файл answer.txt

Затем происходит очистка выделенной памяти и завершение программы.

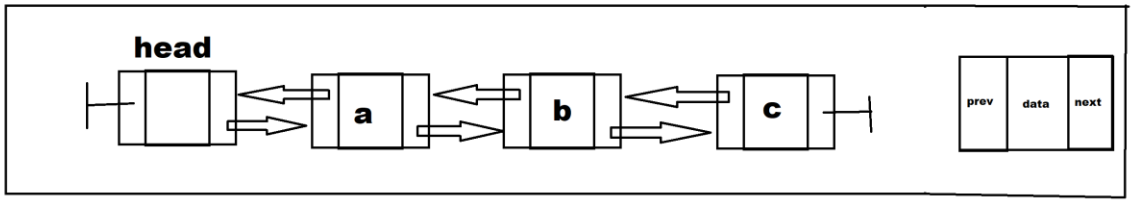
Пример полной работы программы представлен на иллюстрации ниже:

1. Ввод: (a (b ()) c)

2. Преобразование скобочной записи в структуру ИС:



3. Преобразование ИС к ЛС:



4. Вывод ЛС: (a b c)

Полное преобразование:

(a (b ()) c) ⇨ (a b c)

Описание функций.

Структура H_node:

- node* next – указатель на следующий элемент в списке,
- node* prev – указатель на предыдущий элемент в списке,
- node* child - указатель на сына,
- char data - значение
- int isNull – (0 – не пустой элемент, не 0 – пустой элемент)

Структура L_node:

- node* next – указатель на следующий элемент в списке,
- node* prev – указатель на предыдущий элемент в списке,
- char data – значение

Вспомогательные функции:

- char* readsent()

Открывает файл data.txt (если не открылся - выход из функции) и посимвольно записывает содержимое в строку, для которой память выделяется динамически. Закрывает файл и возвращает указатель на строку.

- void shift(char* sent, int ind)

Сдвигает символы в получаемой строке sent ind раз. Необходимо в обработке строк.

- void H_free_list(H_node* elem) и void L_free_list(L_node* elem)

Рекурсивные функции. Принимают первый элемент после головы ИС и ЛС соответственно. Сначала идет проверка, есть ли у элемента сын (только в H_free_list()) и следующий элемент. Если есть, то функция запускает саму себя для него (сына/следующего элемента) и только после этого очищает элемент elem. Из-за этого удаление элементов происходит с конца списка.

- void print_H_list(H_node* lst)

Принимает указатель на голову иерархического списка. Если список пуст выводит пустые скобки и завершает функцию. Если нет, то сын становиться текущим элементом, открывается скобка и пока текущий элемент существует:

- Если у него есть сын, то функция запускает сама себя для этого элемента (она рекурсивна) и ставиться пробел.
- Если элемент не пуст, то выводится поле data, если пуст, то скобки (“()”). Затем пробел.
- Переход к следующему элементу списка.

Скобка закрывается – иерархический список выведен.

- void print_L_list(L_node * lst)

Работает аналогично предыдущей функции, но имеет отличия:

- Открывает файл answer.txt, куда дублирует все выведенное на экран. Файл закрывается в конце.
- Проверяет не сына головы списка, а следующий элемент.
- Не содержит рекурсии из-за отсутствия вложенных списков.

Основные функции:

- void create_H_list(char* s, H_node* lst)

Функция принимает строку s и голову ИС lst. По скобочному выражению из строки s формирует иерархический список. По окончание работы в голове ИС будет находится указатель на ИС. Функция ничего не возвращает.

- node* H_list_to_linar(node* H_lst, node* L_lst, int tabs)

Функция принимает головы ИС и ЛС и кол-во отступов tabs. По элементам ИС, которые имеют не нулевое значение строит ЛС. По окончании работы L_lst (первого вхождения) будет лежать ЛС. Функция возвращает указатель на последний элемент ЛС (необходимо при рекурсии).

- int main()
  1. Запись скобочного выражения в строку (readsent()) Если файл не существовал или строка была пуста, то программа создаст файл “data.txt”, занесет в него пример ИС и заново запустит readsent().
  2. Инициализация голов ИС и ЛС, заполнение их полей.
  3. Создание ИС из строки (create_H_list())
  4. Обработка частного случая “Одиночный атом”.
  5. Вывод ИС на экран (print_H_list())
  6. Преобразование ИС в ЛС (H_list_to_linar())
  7. Вывод ЛС на экран и в файл (print_L_list())
  8. Освобождение памяти:
     a. Строки s
     b. ИС и ЛС, если они не пусты (free_list())

Тестирование.

Тестирование программы представлено в таблице 1.

Таблица 1.

Номер теста:	Входные данные: (файл: “data.txt”)	Выходные данные: (файл: “answer.txt”)
1	`()`	`()`
2	`(a b c)`	`(a b c)`
3	`(a (b c))`	`(a b c)`

4	(a (b c)()(d e f))	(a b c d e f)
5	(() (() ()) ())	()
6	((a) ((b) (c)) (d))	(a b c d)
7	((a b c)(d e f)(g h i))	(a b c d e f g h i)
8	((a (b (c)))(((d) e) f)(g (h) i))	(a b c d e f g h i)
9	a	(a)

Вывод из консоли для тестов будет представлен в папке tests

Выводы.

Были изучены основные понятия иерархического списка и был получен опыт работы с ним на языке программирования Си. Была разработана программа, преобразовывающая иерархический список в линейный.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файлов: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>



char* readsent() {          //функция посимвольного считывания строки из файла
  int size = 10;
  int n = 0;
  char a;
  FILE* f = fopen("data.txt", "r");
  if (!f)
    return NULL;
  char* sent = (char*)malloc(size * sizeof(char));
  while ((sent[n] = fgetc(f)) != EOF ){
    if (sent[n] == ' ')
      continue;
    if (n++ >= size-2) {
      size += 10;
      sent = (char*)realloc(sent, size * sizeof(char));
    }
  }
  sent[n]='\n';
  sent[n+1]='\0';
  fclose(f);
  return sent;
}


void shift(char* sent, int ind) {     //функция сдвига строки-выражения влево
(удаление)
  int i = 0;
  for(i = ind; i<strlen(sent); i++)
    sent[i-ind] = sent[i];
  sent[strlen(sent)-ind]='\0';
}


typedef struct H_node{
  struct H_node* next;   //следующий
  struct H_node* prev;   //предыдущий
  struct H_node* child;  //сын
```

```
  char data;            //данные
  int isNull;           //пустой ли элемент
}H_node;

typedef struct L_node{
  struct L_node* next;
  struct L_node* prev;
  char data;
}L_node;

void H_free_list(H_node* elem){  // функция освобождения памяти, отведенной
под списки
  if (elem->child)
    H_free_list(elem->child);
  if (elem->next)
    H_free_list(elem->next);
  if (elem->child)
    printf("(del)[head]\n");
  else
    printf("(del)[%c]\n",elem->data);
  free(elem);
}

void L_free_list(L_node* elem){
  if (elem->next)
    L_free_list(elem->next);
  printf("(del)[%c]\n",elem->data);
  free(elem);
}

void create_H_list(char* s, H_node* lst){   //функция создания ИС из строки
  char a = '1',b = '1';
  H_node* current = NULL;
  sscanf(s ,"%c%c", &a, &b);

  if((a != '(')&&(a != '\n')){
    lst->child = NULL;
    lst->data = a;
    printf("(single atom)\n");
    return;
  }

  if ((a == '(')&&(b == ')')){
    lst->child = NULL;
    printf("(list is empty)");
    return;
  }
  shift(s,1);
```

```
H_node* elem = (H_node*)malloc(sizeof(H_node));
elem->prev = NULL;
elem->next = NULL;
elem->child = NULL;
elem->data = ' ';
elem->isNull = 0;
printf("(creation of element)(first element of list)");
lst->child = elem;
while (1){
  sscanf(s ,"%c%c", &a, &b);
  if ((a == '(')&&(b == ')')){
    elem->isNull = 1;
    shift(s,2);
  }

  if (a != '(')
    shift(s,1);

  if ((a == '(')&&(b != ')')){
    printf("(moving to the lower level)\n");
    create_H_list(s, elem);
  }

  if ((a != '(')&&(a != ')')){
    elem->data = a;
  }

  if (a == ')'){
    if (elem->prev)
      elem->prev->next = NULL;
    free(elem);
    printf("(deleting an extra element)(sublist finished)\n");

    break;
  }


  current = elem;
  if (!current->child)
    printf("[%c]\n", current->data);
  elem = (H_node*)malloc(sizeof(H_node));
  elem->prev = current;
  elem->next = NULL;
  elem->child = NULL;
  elem->data = ' ';
  elem->isNull = 0;
  printf("(creation of element)");
```

```
    current->next = elem;
  }
}

L_node* H_list_to_linar(H_node* H_lst, L_node* L_lst, int tabs){  // функция
преобразования ИС в ЛС
  H_node* cur_H = H_lst->child;
  L_node* elem = NULL;
  L_node* cur_L = NULL;
  if (!cur_H){
    for(int i=0;i<tabs;i++)printf("\t");
    printf("(list is empty)");
    return NULL;
  }
  if (!L_lst->prev){
    cur_L = (L_node*)malloc(sizeof(L_node));
    for(int i=0;i<tabs;i++)printf("\t");
    printf("(creation of element LL)");
    cur_L->prev = L_lst;
    cur_L->next = NULL;
    cur_L->data = ' ';
    L_lst->next = cur_L;
  }
  else
    cur_L = L_lst;

  while(cur_H){
    printf("\n");
    if (cur_H->child){
      for(int i=0;i<tabs;i++)printf("\t");
      printf("(moving to the lower level of hierarchical list)");
      cur_L = H_list_to_linar(cur_H, cur_L, tabs+1);
      printf("\n");
      for(int i=0;i<tabs+1;i++)printf("\t");
      printf("(hierarchical sublist finished)");
      if (cur_H->next){
        cur_H = cur_H->next;
        continue;
      }
     else
       break;
    }

    cur_L->data = cur_H->data;

    if (cur_H->isNull){
      cur_H = cur_H->next;
      continue;
```

```
    }
    for(int i=0;i<tabs;i++)printf("\t");
    printf("(copying data)[%c]", cur_L->data);

    if (cur_H){
      elem = cur_L;
      cur_L = (L_node*)malloc(sizeof(L_node));
      cur_L->prev = elem;
      cur_L->next = NULL;
      cur_L->data = ' ';
      //for(int i=0;i<tabs;i++)printf("\t");
      printf("(creation of element LL)");
      elem->next = cur_L;
    }
    else
      break;
    cur_H = cur_H->next;

  }
  if (!L_lst->prev){
    cur_L->prev->next = NULL;
    free(cur_L);
    printf("\n(deleting an extra element)(sublist finished)");
  }
  return cur_L;
}

void print_H_list(H_node* lst){     // функция вывода ИС на экран

  if (lst->child == NULL){
    printf("()");
    return;
  }

  H_node* current = lst->child;

  printf("(");
  while (current){
    if (current->child){
      print_H_list(current);
      if (current->next)
        printf(" ");
    }
    else{
      if (!current->isNull)
        printf("%c",current->data);
      else
        printf("()");
```

```
      if (current->next)
        printf(" ");
    }
    current = current->next;
  }
   printf(")");
}

void print_L_list(L_node* lst){           // функция вывода ЛС на экран и в
файл
  FILE* f = fopen("answer.txt", "w");
  if (lst->next == NULL){
    printf("()");
    fprintf(f, "()");
    return;
  }
  L_node* current = lst->next;
  printf("(");
  fprintf(f, "(");
  while (current){
    printf("%c", current->data);
    fprintf(f,"%c", current->data);
    if (current->next){
      printf(" ");
      fprintf(f, " ");
    }
    current = current->next;
  }
  printf(")");
  fprintf(f, ")");

  fclose(f);
}

int main(){
  char* s = readsent();          //считывание строки
  if ((!s)||(s[0]=='\n')){
    printf("The data file could not open, most likely it is missing.\nWe
generously created it in the folder where the program is located.\nWe wrote
down an example there, but you can write your own.");
    FILE* f = fopen("data.txt","w");
    fprintf(f, "(a b(c d) e)");
    fclose(f);
    s = readsent();
  }

  if(s[0]=='\n'){
```

```
}

printf("\n(input)\n%s\n", s);

H_node H_lst;
H_lst.prev = NULL;
H_lst.next = NULL;
H_lst.child = NULL;
H_lst.data = ' ';

L_node L_lst;
L_lst.prev = NULL;
L_lst.next = NULL;

create_H_list(s, &H_lst);      //считывания ИЛ

if (H_lst.data != ' '){       // Одиночный атом
  L_lst.data = H_lst.data;
  printf("\n(linear list)\n(%c)\n",L_lst.data);
  FILE* f = fopen("answer.txt","w");
  fprintf(f, "(%c)", L_lst.data);
  fclose(f);
  return 0;
}

printf("\n\n");

printf("(hierarchical list)\n");
print_H_list(&H_lst);          // вывести на экран ИС

printf("\n\n");

H_list_to_linar(&H_lst, &L_lst,0);    //преобразовать ИС в ЛС

printf("\n\n");

printf("(linear list)\n");
print_L_list(&L_lst);               // вывести на экран и в файл ЛС

free(s);
printf("\n\n");
if (H_lst.child){
  printf("(deleting of hierarchical list)\n");
  H_free_list(H_lst.child);
}
printf("\n");
if (L_lst.next){
  printf("(deleting of linear list)\n");
```

```
    L_free_list(L_lst.next);
  }
  return 0;
}
```