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.

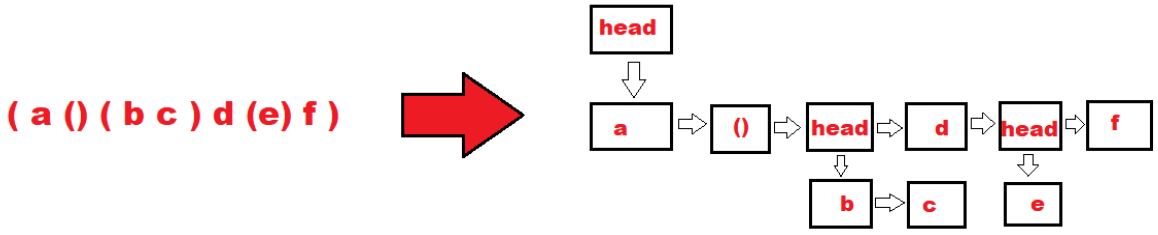
По заданию необходимо преобразовать иерархический список в линейный.

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.

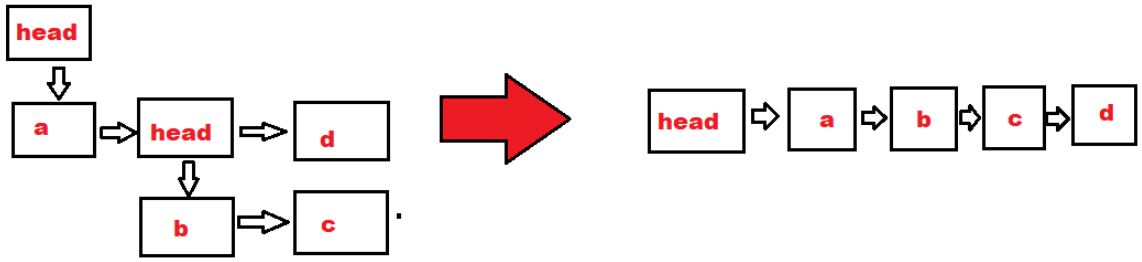
Пример работы функции считывания:



Затем нужно преобразовать иерархический список в линейный, то есть если у элемента есть сын, то он (сын) становиться следующим

элементом для предшествующего родителя элемента, который имел какое-либо значение (т.е. он не должен быть родителем или должен быть головой иерархического списка). Последний же элемента должен ссылаться на следующий элемент после родителя подсписка.

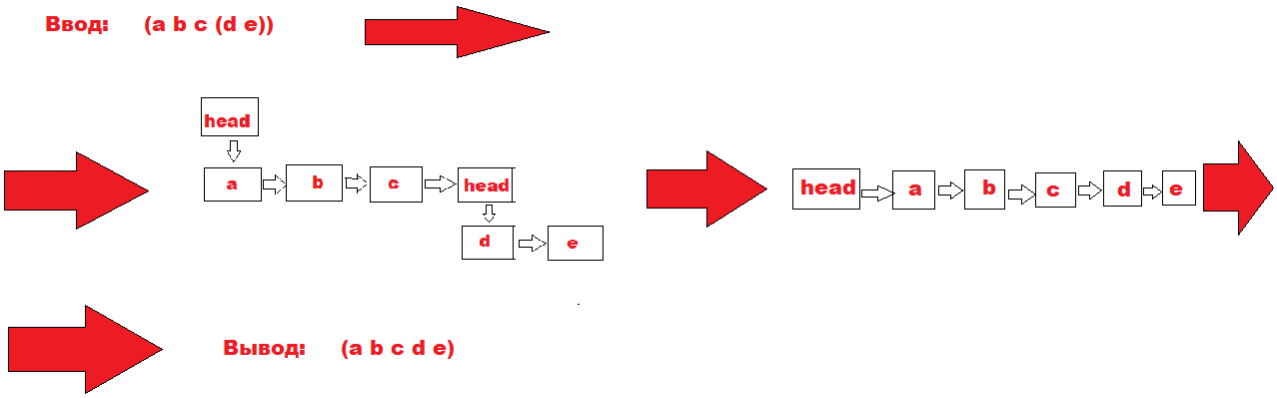
Пример работы функции преобразования:



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:



Выполнение работы.

Структура 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)

Вспомогательные функции:

- Char* readsent()

Открывает файл data.txt 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 ind раз. Необходимо в обработке строк.

- Void free_list(node* elem)

Рекурсивная функция. Принимает первый элемент после головы списка. Подходит для очистки памяти и линейного, и иерархического списка. Сначала идет проверка, есть ли у элемента сын 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(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(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, node* lst)

Функция принимает строку со скобочным выражением и голову пустого списка. Инициализируются 2 символьные переменные для обработки выражения и указатель на текущий элемент списка. Если

первыми элементами идут скобки (“()”), то список пуст, ставиться соответствующее значение в поле child и функция завершается.

Иначе удаляется 1 символ из строки (“(”) функцией shift и создается и инициализируется новый элемент, который становится сыном головы списка.

Затем цикл while(1):

- Если встречается “()”, то элемент пуст, что ставиться в поле isNull как 1 (это нужно для функций вывода списков на экран).
- Если символ не скобки, то он записывается в поле data.
- Если встречается открывающаяся скобка, то функция запускает саму себя еще раз для текущего элемента elem.
- Если встречается закрывающаяся скобка, то это знак конца списка. Если существует предыдущий элемент, то мы обнуляем его указатель на этот элемент и очищаем этот. Это необходимо потому, что во время считывания последнего элемента сразу создается следующий, в который уже ничего не попадёт, поэтому его нужно удалить.
- Запоминается указатель на текущий элемент, текущий элемент заменяется на новый и в старый элемент вносится указатель на новый на новый элемент.

Обработка строки осуществляется с помощью sscanf() и shift(). sscanf() считывает символы из строки s, а shift() удаляет обработанные символы.

После совершённого значительного действия в консоль пишутся системные сообщения. В файл answer.txt они не попадут.

- Node* H_list_to_linar(node* H_lst, node* L_lst)

Получаем голову на иерархический и линейный список. Инициализация указателя на текущий элемент в иерархическом списке (сын головы иерархического списка), на новый элемент и на элемент линейного списка.

Если иерархический список пуст, то функция завершается.

Если это первый элемент линейного списка (не считая голову), то выделяем на него память, проставляем значения в полях и вставляем новый элемент в линейный список. Если не первый, то текущий элемент равен голове списка. Это нужно, чтобы предыдущий элемент ИС указывал на сына ТЭИС.

Пока существует текущий элемент иерархического списка:

- Если у текущего элемента иерархического списка (ТЭИС) есть сын, то в текущий элемент линейного списка (ТЭЛС) становиться последним элементом вложенного списка (списка, начинающегося с ТЭИС). Это значение возвращает H_list_to_linar() (рекурсивная функция), в которую мы отправляем ТЭИС и ТЭЛС. Затем если существует следующий элемент ТЭИС, то ТЭИС меняется на него и переход на следующую итерацию, иначе выход из цикла.
- Если ТЭИС не пуст, то происходит перенос данных в ТЭЛС.
- Перенос данных из ТЭИС в ТЭЛС.
- Если ТЭИС существует, то запоминается указатель на ТЭЛС, ТЭЛС заменяется на новый и в старый элемент ЛС

ставиться указатель на новый элемент ЛС. Иначе – выход из цикла.

- Переход к следующему ТЭИС

Если это первый заход в функцию (у головы ЛС не предыдущего члена), то мы удаляем последний лишний элемент линейного списка по той же причине, что и в функции create_H_list().

Функция возвращает указатель на последний элемент списка.

После совершённого значительного действия в консоль пишутся системные сообщения. В файл answer.txt они не попадут.

- Int main()
  1. Запись скобочного выражения в строку (readsent())
  2. Инициализация голов ИС и ЛС, заполнение их полей.
  3. Создание ИС из строки (create_H_list())
  4. Вывод ИС на экран (print_H_list())
  5. Преобразование ИС в ЛС (H_list_to_linar())
  6. Вывод ЛС на экран и в файл (print_L_list())
  7. 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)`

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");
  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 node{
  struct node* next;    //следующий
  struct node* prev;    //предыдущий
  struct node* child;   //сын
  char data;            //данные
  int isNull;           //пустой ли элемент
```

```
}node;

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

void create_H_list(char* s, node* lst){    //функция создания ИС из
строки
  char a = '1',b = '1';
  node* current = NULL;
  sscanf(s ,"%c%c", &a, &b);
  if ((a == '(')&&(b == ')')){
    lst->child = NULL;
    printf("(list is empty)");
    return;
  }
  shift(s,1);

  node* elem = (node*)malloc(sizeof(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 = (node*)malloc(sizeof(node));
    elem->prev = current;
    elem->next = NULL;
    elem->child = NULL;
    elem->data = ' ';
    elem->isNull = 0;
    printf("(creation of element)");
    current->next = elem;
  }
}

node* H_list_to_linar(node* H_lst, node* L_lst){   // функция
преобразования ИС в ЛС
  node* cur_H = H_lst->child;
  node* elem = NULL;
  node* cur_L = NULL;
  if (!cur_H){
    printf("(list is empty)");
    return NULL;
  }
  if (!L_lst->prev){
    cur_L = (node*)malloc(sizeof(node));
    printf("(creation of element)");
    cur_L->prev = L_lst;
    cur_L->next = NULL;
    cur_L->child = NULL;
    cur_L->data = ' ';
    cur_L->isNull = 0;
    L_lst->next = cur_L;
  }
  else
    cur_L = L_lst;
```

```
  while(cur_H){

    if (cur_H->child){
      cur_L = H_list_to_linar(cur_H, cur_L);

      if (cur_H->next){
        cur_H = cur_H->next;
        continue;
      }
     else
       break;
    }

    cur_L->data = cur_H->data;
    if (cur_H->isNull)
      cur_L->isNull = 1;
    printf("(creation of element)[%c]\n", cur_L->data);

    if (cur_H){
      elem = cur_L;
      cur_L = (node*)malloc(sizeof(node));
      cur_L->prev = elem;
      cur_L->next = NULL;
      cur_L->child = NULL;
      cur_L->data = ' ';
      cur_L->isNull = 0;
      printf("(creation of element)");
      elem->next = cur_L;
    }
    else
      break;
    cur_H = cur_H->next;

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

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

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

```
  }

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

```
    current = current->next;
  }
  printf(")");
  fprintf(f, ")");

  fclose(f);
}

int main(){
  char* s = readsent();          //считывание строки
  printf("\n(input)\n%s\n", s);

  node H_lst;
  H_lst.prev = NULL;
  H_lst.next = NULL;
  H_lst.child = NULL;

  node L_lst;
  L_lst.prev = NULL;
  L_lst.next = NULL;
  L_lst.child = NULL;

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

  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);    //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");
    free_list(H_lst.child);
  }
  printf("\n");
  if (L_lst.next){
    printf("(deleting of linear list)\n");
    free_list(L_lst.next);
```

```
    }
    return 0;
}
```