Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ОТЧЕТ

по разминке №3

Тема работы: Крючки

Выполнил

студент: гр. 251003 Панкратьев Е.С.

Проверил: Фадеева Е.П.

Минск 2023

Содержание

[1 Постановка задачи 3](#_Toc136781412)

[2 Дополненная постановка задачи 4](#_Toc136781413)

[3 Метод решения 5](#_Toc136781414)

[3.1 Описание оператора try..except..end 5](#_Toc136781415)

[3.2 Условия ввода 5](#_Toc136781416)

[3.3 Проверка введенных данных 5](#_Toc136781417)

[3.4 Краткое описание алгоритма решения задачи 5](#_Toc136781418)

[4 Описание алгоритмов решения задачи 7](#_Toc136781419)

[5 Структура данных 8](#_Toc136781420)

[6 Результаты расчетов 10](#_Toc136781421)

[Приложение А 11](#_Toc136781422)

[Приложение Б 14](#_Toc136781423)

[Приложение В 17](#_Toc136781424)

# Постановка задачи

Вводится количество крючков, из которых можно сделать «и» и «ш». Найти количество последовательностей данных символов, которые можно составить из данного количества крючков.

# Дополненная постановка задачи

Вводится количество крючков, из которых можно сделать «и» и «ш». Найти количество последовательностей данных символов и вывести сами последовательности, которые можно составить из данного количества крючков.

# Метод решения

## Описание оператора try..except..end

Оператор try..except..end имеет вид:

try  
 операторы;  
except  
  блок обработки исключений;  
end;

Выполнение блока начинается с секции try, при отсутствии исключительных ситуаций только она и выполняется. Секция except получает управление в случае возникновения исключительной ситуации. После обработки выполняются операторы, стоящие после end.

## Условия ввода

Условия ввода:

* количество крючков должно быть целочисленным значением;
* количество крючков должно быть в диапозоне [2..1000];

## Проверка введенных данных

Проверка введенных данных происходит с помощью цикла с предусловием repeat..until, чтобы при вводе некорректных данных пользователь заново заполнял их. В теле цикла с помощью оператора try..except..end (описание оператора см. [главу 3.1](#_Описание_оператора_try..except..end)) проверяем соответствие с типом данным; оператором if проверяет принадлежность заданному диапазону, где это необходимо.

## Краткое описание алгоритма решения задачи

Алгоритм решения задачи по поиску всех возможных последовательностей крючков:

Таблица 1 – Краткое описание алгоритма решения задачи

|  |  |
| --- | --- |
| Шаг | Действия |
| Базовый случай. Если количество оставшихся крючков равно 0 | Вывести текущую последовательность.  величить счетчик допустимых последовательностей |

Продолжение Таблицы 1

|  |  |
| --- | --- |
| Итерационный шаг. Если стек не пустой | Уменьшив количество оставшихся крючков на соответствующее значение.  Добавить соответствующий символ к текущей последовательности.  Добавить на вершину стека новый элемент |
| Поиск последовательностей | Для каждого возможного значения количества крючков извлекать элементы из стека и обрабатывать их |
| Вывод результатов | Вывести все найденные допустимые последовательности.  Вывести общее количество допустимых последовательностей. |

Алгоритм основан на использовании стека для хранения всех возможных промежуточных состояний. На каждом шаге итерации извлекается элемент из стека и проверяется количество оставшихся крючков. Если количество крючков достаточно для добавления символов 'ш' или 'и', создается новый элемент соответствующего количества уменьшенных крючков и добавленными символами в последовательность. Этот новый элемент помещается в стек для дальнейшей обработки. Если количество оставшихся крючков равно 0, то достигнут базовый случай, и текущая последовательность выводится. После завершения поиска всех комбинаций выводятся результаты – найденные последовательности и общее количество допустимых последовательностей.

# Описание алгоритмов решения задачи

Таблица 2 – Описание алгоритмов решения задачи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п.п. | Наименование алгоритма | Название алгоритма | Формальные  параметры | Предпо-лагаемый тип реали-зации |
| 1 | Основной алгоритм | Вызов следующих подпрограмм:  OutputAllHooks и GetAmountOfHooks |  |  |
| 2 | OutputAllHooks  (  AHooksAmount,  ACount  ) | Находит количество возможных последовательностей букв и саму последовательность из AHooksAmount крючков. Количество записывает в ACount | AHooksAmount – получает от фактического параметра значение.  ACount – получает от фактического параметра адрес | Проце-дура |
| 3 | GetAmountOf-Hooks  (  Res  ) | Получение положительного целочисленного числа от пользователя. Результат записывает в Res | Res – получает от фактического параметра адрес,  возвращаемый параметр | Функция. Res – возвращаемый функцией параметр |
| 4 | Push  (  AStack,  AItem  ) | Добавляет на вершину стека AStack элемент AItem | AStack – получает от фактического параметра адрес.  AItem – получает от фактического параметра адрес с защитой | Проце-дура |
| 5 | Pop  (  AStack,  Res  ) | Извлекает из вершина стека AStack элемент и записывает его в Res | AStack – получает от фактического параметра адрес.  Res – получает от фактического параметра адрес,  возвращаемый параметр | Функция. Res – возвращаемый функцией параметр |

# Структура данных

Таблица 3 – Структура типов программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| TStack | Record  FTop: PItem;  FCount: Integer;  End | Представляет стек, хранящий указатель на первый элемент и количество элементов в стеке |
| PStack | ^TStack | Указатель на стек |
| TStackItem | Record  AmountHooks: Integer;  Sequence: String;  End; | Представляет данные стека для нахождения последовательностей букв. |
| TItem | Record  FData: TStackItem;  FNext: PItem;  End; | Представляет элемент стека |
| PItem | ^TItem | Указатель на элемент стека |

Таблица 4 – Структура данных основной программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| Count | Integer | Количество последовательностей |
| HooksAmount | Integer | Количество крючков |

Таблица 5 – Структура данных алгоритма OutputAllHooks(AHooksAmount, ACount)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| AHooks-  Amount | Integer | Количество крючков | Формальный |
| ACount | Integer | Количество последовательностей | Формальный |
| Stack | TStack | Хранения промежуточных состояний при поиске всех возможных последовательностей крючков | Локальный |
| Item | TStackItem | Текущий элемент стека | Локальный |
| NewItem | TStackItem | Новый элемент стека | Локальный |

Таблица 6 – Структура данных алгоритма GetAmountOfHooks(Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Res | Integer | Введенное количество крючков | Формальный |
| IsCorrect | Boolean | Индикатор правильного ответа | Локальный |
| Min | Integer | Минимальное количество крючков | Локальный |
| Max | Integer | Максимальное количество крючков | Локальный |

Таблица 7 – Структура данных алгоритма Push(AStack, AItem)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| AStack | PStack | Стек, в который необходимо добавить новый элемент | Формальный |
| AItem | TStackItem | Данные, которые необходимо добавить | Формальный |
| NewItem | PItem | Новый элемент стека | Локальный |

Таблица 8 – Структура данных алгоритма Pop(AStack, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| AStack | PStack | Стек, в котором необходимо извлечь элемент | Формальный |
| Res | TStackItem | Извлеченные данные | Формальный |
| Item | PItem | Извлеченный элемент стека | Локальный |

# Результаты расчетов

В ходе работы были получены следующие результаты:

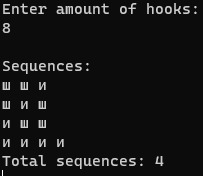


Рисунок 1 – Результаты расчетов

Приложение А

(обязательное)

Исходный код программы

program Iterative;

{

Finds the whole sequence of hooks

Finds the entire hook sequence. "и" gives 2 hooks, "ш" - 3

hooks

}

uses

System.SysUtils, uStack in 'uStack.pas';

// Finds the whole sequence of hooks

procedure OutputAllHooks(AHooksAmount: Integer; var ACount:

Integer);

type

TStackItem = record

AmountHooks: Integer;

Sequence: string;

end;

// TStackItem - stack element

var

Stack: TStack<TStackItem>;

Item: TStackItem;

NewItem: TStackItem;

// Stack - store the intermediate states

// Item - current stack item being processed

// NewItem - new stack item to be pushed

begin

// Create the stack

Stack := TStack<TStackItem>.Create;

// Initialization for the first iteration

NewItem.AmountHooks := AHooksAmount;

NewItem.Sequence := '';

Stack.Push(NewItem);

// Finding all sequences

while Stack.Count > 0 do

begin

Item := Stack.Pop;

if Item.AmountHooks >= 3 then

begin

// Decrease the remaining hooks by 3

NewItem.AmountHooks := Item.AmountHooks - 3;

// Append 'ш' to the sequence

NewItem.Sequence := Item.Sequence + 'ш ';

// Push the new item to the stack for further processing

Stack.Push(NewItem);

end;

if Item.AmountHooks >= 2 then

begin

// Decrease the remaining hooks by 2

NewItem.AmountHooks := Item.AmountHooks - 2;

// Append 'и' to the sequence

NewItem.Sequence := Item.Sequence + 'и ';

// Push the new item to the stack for further processing

Stack.Push(NewItem);

end

else if Item.AmountHooks = 0 then

begin

// Output the sequence if there are no more hooks

Writeln(Item.Sequence);

// Increment the count of valid sequences

Inc(ACount);

end;

end;

// Destroy the stack

Stack.Destroy;

end;

// Returns the correct number of hooks

function GetAmountOfHooks: Integer;

const

Min = 2;

Max = 1000;

// Maximum and minimum allowed value for the number of hooks

var

IsCorrect: Boolean;

// Flag to indicate if the input is correct

begin

Writeln('Enter amount of hooks: ');

// Repeat the loop until the input is correct

repeat

IsCorrect := True;

try

// Read the user input and assign it to the function result

Readln(Result);

except

// Output an error message for invalid input

IsCorrect := False;

Writeln('Incorrect input');

end;

if ((Result < Min) or (Result > Max)) and IsCorrect then

begin

IsCorrect := False;

// Output an error message for invalid input

Writeln('Incorrect input');

end;

until IsCorrect;

Writeln;

end;

var

Count: Integer;

HooksAmount : Integer;

begin

// Initialize the count of valid sequences

Count:= 0;

// Finds the whole sequence of hooks

HooksAmount := GetAmountOfHooks;

Writeln('Sequences:');

OutputAllHooks(HooksAmount, Count);

// Output sequence count

Writeln('Total sequences: ', count);

Readln;

end.

Приложение Б

(обязательное)

Исходный код программы (модуль uStack)

unit uStack;

interface

type

// Stack

TStack<T> = class

private type

PItem = ^TItem;

TItem = record

FData: T;

FNext: PItem;

end;

// Item - stack element

private

FTop: PItem;

FCount: Integer;

// FTop - top of the stack

// FCount - amount of items in the stack

public

constructor Create;

destructor Destroy; override;

// Pushes an item onto the stack

procedure Push(const AItem: T);

// Clears the stack

procedure Clear;

// Pops an item from the stack

function Pop: T;

// Returns the item at the top of the stack without

// removing it

function Peek: T;

// Number of items in the stack

property Count: Integer read FCount;

end;

implementation

constructor TStack<T>.Create;

begin

// Initialize top of the stack to nil

FTop := nil;

// Initialize count to 0

FCount := 0;

end;

destructor TStack<T>.Destroy;

begin

// Clear the stack before destroying the object

Clear;

inherited;

end;

procedure TStack<T>.Clear;

var

I: Integer;

begin

// Pop each item from the stack to clear it

for I := FCount - 1 downto 0 do

Pop;

end;

procedure TStack<T>.Push(const AItem: T);

var

NewItem: PItem;

begin

// Create a new item for the stack

New(NewItem);

// Set the new item as the top of the stack

NewItem^.FData := AItem;

NewItem^.FNext := FTop;

FTop := NewItem;

// Increase the count of items in the stack

Inc(FCount);

end;

function TStack<T>.Pop: T;

var

Item: PItem;

begin

// Get the top item from the stack

Item := FTop;

FTop := FTop^.FNext;

// Get the data from the top item

Result := Item^.FData;

// Free the memory allocated for the top item

Dispose(Item);

// Decrease the count of items in the stack

Dec(FCount);

end;

function TStack<T>.Peek: T;

begin

// Return the data from the top item without removing it

Result := FTop^.FData;

end;

end.

Приложение В

(обязательное)

Тестовые наборы

**Тестовая ситуация: некорректный ввод данных**

**Тест 1**

Исходные данные: Некорректный ввод дат

Ожидаемый результат: Повторная попытка

Полученный результат:

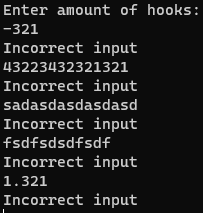


Рисунок 2 – Результаты расчетов

**Тестовая ситуация: корректный ввод данных**

Таблица 9 – Тестовые наборы

|  |  |  |
| --- | --- | --- |
| Тест | Рекурсивный вариант | Итерационный вариант |
| 1. |  |  |
| 2. |  |  |

Продолжение Таблицы 9

|  |  |  |
| --- | --- | --- |
| 3. |  |  |
| 4. |  |  |
| 5. |  |  |

Продолжение Таблицы 9

|  |  |  |
| --- | --- | --- |
| 6. |  |  |