**SPRAWOZDANIE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Przedmiot** | Algorytmy i struktury danych | **Zadanie** | 4\_02 |
| **Autor** | Kamil Borkowski | **Grupa** | WCY22IY1S1 |
| **Temat** | Dodawanie dużych liczb na stosach | | |

1. Treść

Zaimplementuj algorytm dodający dwie liczby za pomocą stosów

Metoda realizacji

Produkty dodawania przechowamy w odrębnych stosach i będziemy je do siebie dodawać zaczynając od liczby jedności.

* 1. Założenia / ograniczenia dotyczące danych:
     1. Dane wejściowe *(opis)*

Pierwsza liczba – n1– wczytywana z klawiatury

Druga liczba – n2 – wczytywana z klawiatury

* + 1. Dane wyjściowe *(opis)*

Suma liczb – wyświetlone na ekranie

1. Realizacja
   1. Algorytm *(lista kroków lub schemat blokowy)*

Deklaracja struktury listy dla liczb i wyniku.

Wpisanie liczb do przynależnych im stosów

Dodawanie kolejnych cyfr liczb zaczynając od jedności

Wyświetlenie stosu sumy

* 1. Kod źródłowy

//================================================

//Zadanie 4\_02 jezyk C++

//Dodawanie duzych lich przy wykorzystaniu stosu

//WCY21IY1S1 Borkowski Kamil

//================================================

#include <iostream>

#include <cstdlib>

using namespace std;

struct val\_POM

{

int value;

val\_POM\* prev;

};

struct val\_ONE

{

int value;

val\_ONE\* prev;

};

struct val\_TWO

{

int value;

val\_TWO\* prev;

};

struct val\_THREE

{

int value;

val\_THREE\* prev;

};

int main()

{

int n1,n2,n3,over=0;

cout<<"Podaj pierwsza liczbe dodawania"<<endl;

cin>>n1;

cout<<"Podaj druga liczbe dodawania"<<endl;

cin>>n2;

//umieszczanie cyfr pierwszej liczby na stosie pomocniczym(w odwrotnej kolejnosci niz chciana)

val\_POM\* head = NULL; //pierwszy element listy pomocniczej, poczatkowo pusty

val\_POM\* current = head; //pomocniczy element listy, sluzacy do poruszania sie po liscie

current = head;

while(n1>0)

{

val\_POM\* new\_node = (val\_POM\*)malloc(sizeof(val\_POM));

new\_node->value = n1%10;

n1=n1/10;

new\_node->prev = head;

head = new\_node;

}

//wpisanie liczb z stosu pom do val\_one (pierwszy element stosu pomocniczego bedzie ostatnim elementem stosu val\_one)

current = head;

val\_ONE\* head1 = NULL; //pierwszy element listy dla pierwszej liczby, poczatkowo pusty

val\_ONE\* current1 = head1; //pomocniczy element listy, sluzacy do poruszania sie po liscie

current1 = head1;

while(current)

{

val\_ONE\* new\_node = (val\_ONE\*)malloc(sizeof(val\_ONE));

new\_node->value=current->value;

new\_node->prev = head1;

head1 = new\_node;

current = current->prev;

}

cout<<"Stos pierwszy - od lewej gora stosu:"<<endl;

current1 = head1;

while(current1 != NULL)

{

cout << current1->value<<'\t';

current1 = current1->prev;

}

cout<<endl;

//zwalniamy pamiec stosu pomocniczego

current = head;

while(current != NULL)

{

val\_POM\* pom = current;

current = current->prev;

free(pom);

}

//umieszczanie cyfr drugiej liczby na stosie pomocniczym(w odwrotnej kolejnosci niz chciana)

head = NULL; //pierwszy element listy pomocniczej, poczatkowo pusty

current = head; //pomocniczy element listy, sluzacy do poruszania sie po liscie

while(n2>0)

{

val\_POM\* new\_node = (val\_POM\*)malloc(sizeof(val\_POM));

new\_node->value = n2%10;

n2=n2/10;

new\_node->prev = head;

head = new\_node;

}

//wpisanie liczb z stosu pom do val\_two (pierwszy element stosu pomocniczego bedzie ostatnim elementem stosu val\_two)

current = head;

val\_TWO\* head2 = NULL; //pierwszy element listy dla drugiej liczby, poczatkowo pusty

val\_TWO\* current2 = head2; //pomocniczy element listy, sluzacy do poruszania sie po liscie

current2 = head2;

while(current)

{

val\_TWO\* new\_node = (val\_TWO\*)malloc(sizeof(val\_TWO));

new\_node->value=current->value;

new\_node->prev = head2;

head2 = new\_node;

current = current->prev;

}

cout<<"Stos drugi - od lewej gora stosu"<<endl;

current2 = head2;

while(current2 != NULL)

{

cout << current2->value<<'\t';

current2 = current2->prev;

}

cout<<endl;

//uzupelnianie stosu sumy

val\_THREE\* head3 = NULL; //pierwszy element listy, poczatkowo pusty

val\_THREE\* current3 = head3; //pomocniczy element listy, sluzacy do poruszania sie po liscie

current1 = head1;

current2 = head2;

while((current1)||(current2)||over)

{

val\_THREE\* new\_node = (val\_THREE\*)malloc(sizeof(val\_THREE));

new\_node->value = ((current1 ? current1->value : 0) + (current2 ? current2->value : 0)+over)%10;//jezeli skonczy nam sie jeden stos to dodajemy 0

if ((current1 ? current1->value : 0) + (current2 ? current2->value : 0)+over >= 10) over = 1;//jezeli dodawanie da wiecej niz 9 to w nastepnym sumowaniu dodamy +1

else over=0;

new\_node->prev = head3;

head3 = new\_node;

if (current1) current1 = current1->prev;

if (current2) current2 = current2->prev;

}

cout<<"Wynik dodawania: "<<endl;

current3 = head3;

while(current3 != NULL)

{

cout << current3->value;

current3 = current3->prev;

}

//=========czyszczenie pamieci================

current = head;

while(current != NULL)

{

val\_POM\* pom = current;

current = current->prev;

free(pom);//zwolniamy pammiec

}

current1 = head1;

while(current1 != NULL)

{

val\_ONE\* pom = current1;

current1 = current1->prev;

free(pom);//zwolniamy pammiec

}

current2 = head2;

while(current2 != NULL)

{

val\_TWO\* pom = current2;

current2 = current2->prev;

free(pom);//zwolniamy pammiec

}

current3 = head3;

while(current3 != NULL)

{

val\_THREE\* pom = current3;

current3 = current3->prev;

free(pom);//zwolniamy pammiec

}

return 0;

}

* 1. Dane wejściowe *(ekran uruchomienia programu)*

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Oprogramowanie graficzne

Opis wygenerowany automatycznie

* 1. Dane wyjściowe *(ekran uruchomienia programu)*

Obraz zawierający tekst, elektronika, zrzut ekranu, wyświetlacz

Opis wygenerowany automatycznie

1. Wnioski *(złożoność asymptotyczna algorytmu przy użyciu notacji* ***O lub innej i inne wnioski)***

Złożoność obliczeniowa programu wynosi: O(n), zależna od długości najdłuższej liczby.