## Міністерство освіти і науки України

## ФАКУЛЬТЕТ ІНФОРМАТИКИ

КАФЕДРА

#### Реєстраційний №\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## КУРСОВА РОБОТА

**Тема: Програмний комплекс для роботи (розробки) візитних карток**

#### Рекомендована до захисту

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2008р.

Робота захищена

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2008р.

з оцінкою

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Підписи членів комісії

Зміст

# Вступ

# Теорія

# Практична частина

# Висновки

# Література

**Вступ**

# Borland C++ Builder - запропоноване недавно компанією Borland засіб швидкої розробки програм, що дозволяє створювати програми мовою C++, використовуючи при цьому середовище розробки і бібліотеку компонентів Delphi. Надалі в теоретичній частині буде розглядатися середовище розробки C++ Builder і основні прийоми, застосовувані при проектуванні користувацького інтерфейсу для обробки візитних карток.

C++ Builder являє собою SDI-додаток, головне вікно якого містить інструментальну панель, що набудовується, (ліворуч) і палітру компонентів (праворуч). Крім цього, за замовчуванням при запуску C++ Builder з'являються вікно інспектора об'єктів (ліворуч) і форма нового програми (праворуч). Під вікном форми програми знаходиться вікно редактора коду.

**Теорія**

### Засоби організації збереження і обробки даних для інтерфейсних

### програм

#### 

#### **Методи організації і збереження лінійних списків**

Лінійний список - це кінцева послідовність однотипних елементів (вузлів), можливо, з повтореннями. Кількість елементів у послідовності називається довжиною списку, причому довжина в процесі роботи програми може змінюватися.

Лінійний список F, що складається з елементів D1,D2,...,Dn, записують у виді послідовності значень укладеної в кутові дужки F=, або представляють графічно (див.мал.12).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D1 | **** | D2 | **** | D3 | **** | ... | **** | Dn |
| **Рис.12. Зображення лінійного списку.** | | | | | | | | |

Наприклад, F1=<2,3,1>,F2=<7,7,7,2,1,12>, F3=<>. Довжина списків F1, F2, F3 дорівнює відповідно 3,6,0.

При роботі зі списками на практиці найчастіше приходиться виконувати наступні операції:

- знайти елемент із заданою властивістю;

- визначити перший елемент у лінійному списку;

- уставити додатковий елемент до або після зазначеного вузла;

- виключити визначений елемент зі списку;

- упорядкувати вузли лінійного списку у визначеному порядку.

У реальних мовах програмування немає якої-небудь структури даних для представлення лінійного списку так, щоб усі зазначені операції над ним виконувалися в однаковому ступені ефективно. Тому при роботі з лінійними списками важливим є представлення використовуваних у програмі лінійних списків таким чином, щоб була забезпечена максимальна ефективність і за часом виконання програми, і по обсязі необхідної пам'яті.

Методи збереження лінійних списків розділяються на методи послідовного і зв'язаного збереження. Розглянемо найпростіші варіанти цих методів для списку з цілими значеннями F=<7,10>.

При послідовному збереженні елементи лінійного списку розміщаються в масиві d фіксованих розмірів, наприклад, 100, і довжина списку вказується в перемінної l, тобто в програмі необхідно мати оголошення виду

float d[100]; int l;

Розмір масиву 100 обмежує максимальні розміри лінійного списку. Список F у масиві d формується так:

d[0]=7; d[1]=10; l=2;

Отриманий список зберігається в пам'яті відповідно до схеми на мал.13.

|  |  |
| --- | --- |
| l: | 2 |
| d: | 7 | 10 |  |  | ... |  |  |
|  | [0] | [1] | [2] | [3] |  | [98] | [99] |
| **Рис.1. Послідовне збереження лінійного списку.** | | | | | | | |

При зв'язаному збереженні як елементи збереження використовуються структури, зв'язані по одній з компонентів у ланцюжок, на початок якої (першу структуру) указує покажчик dl. Структура утворюючий елемент збереження, повинна крім відповідного елемента списку містити і покажчик на сусідній елемент збереження.

Опис структури і покажчика в цьому випадку може мати вид:

typedef struct snd /\* структура елемента збереження \*/

{ float val; /\* елемент списку \*/

struct snd \*n ; /\* покажчик на елемент збереження \*/

} DL;

DL \*p; /\* покажчик поточного елемента \*/

DL \*dl; /\* покажчик на початок списку \*/

Для виділення пам'яті під елементи збереження необхідно користуватися функцією malloc(sizeof(DL)) або calloc(l,sizeof(DL)). Формування списку в зв'язаному збереженні може здійснюється операторами:

p=malloc(sizeof(DL));

p->val=10; p->n=NULL;

dl=malloc(sizeof(DL));

dl->val=7; dl->n=p;

В останньому елементі збереження (кінець списку) покажчик на сусідній елемент має значення NULL. Одержуваний список зображений на мал.2.



**Рис.2. Зв'язне збереження лінійного списку.**

#### 

#### Операції зі списками при послідовному збереженні

При виборі методу збереження лінійного списку варто враховувати, які операції будуть виконуватися і з якою частотою, час їхнього виконання й обсяг пам'яті, необхідний для збереження списку.

Нехай мається лінійний список з цілими значеннями і для його збереження використовується масив d (з числом елементів 100), а кількість елементів у списку вказується перемінної l. Реалізація зазначених раніше операцій над списком представляється наступними фрагментами програм які використовують оголошення:

float d[100];

int i,j,l;

1) печатка значення першого елемента (вузла)

if (і<0 || і>l) printf("\n немає елемента");

else printf("d[%d]=%f ",і,d[і]);

2) видалення елемента, що випливає за i-тым вузлом

if (і>=l) printf("\n немає наступного ");

l--;

for (j=і+1;j<="1" if вузла i-того сусідів обох печатка 3) d[j]="d[j+1];">=l) printf("\n немає сусіда");

else printf("\n %d %d",d[і-1],d[і+1]);

4) додавання нового елемента new за i-тым вузлом

if (і==l || і>l) printf("\n не можна додати");

else

{ for (j=l; j>i+1; j--) d[j+1]=d[j];

d[i+1]=new; l++;

}

5) часткове упорядкування списку з елементами ДО1,ДО2,...,Кl у

список K1',K2',...,Ks,K1,Kt",...,Kt", s+t+1=l так, щоб K1'= K1.

{ int t=1;

float aux;

for (i=2; i<=l; i++) if (d[i]=2; j--) d[j]=d[j-1];

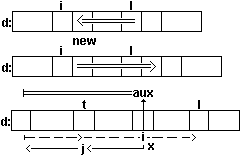
t++;

d[i]=aux;

}

}

Схема руху індексів i,j,t і значення aux=d[i] при виконанні приведеного фрагмента програми приведена на мал.3.



**Рис.3. Рух індексів при виконанні операцій над списком у послідовному збереженні.**

Кількість дій Q, необхідних для виконання приведених операцій над списком, визначається співвідношеннями: для операцій 1 і 2 - Q=1; для операцій 3,4 - Q=l; для операції 5 - Q=l\*l.

Помітимо, що взагалі операцію 5 можна виконати при кількості дій порядку l, а операції 3 і 4 для включення і виключення елементів наприкінці списку, що часто зустрічаються при роботі зі стеками, - при кількості дій 1.

Більш складна організація операцій потрібно при розміщенні в масиві d декількох списків, або при розміщенні списку без прив'язки його початку до першого елемента масиву.

#### *Операції зі списками при зв'язному збереженні*

При простому зв'язаному збереженні кожен елемент списку являє собою структуру nd, що складається з двох елементів: val - призначений для збереження елемента списку, n - для покажчика на структуру, що містить наступний елемент списку. На перший елемент списку вказує покажчик dl. Для всіх операцій над списком використовується опис:

typedef struct nd

{ float val;

struct nd \* n; } ND;

int i,j;

ND \* dl, \* r, \* p;

Для реалізації операцій можуть використовуватися наступні фрагменти програм:

1) печатка значення i-го елемента

r=dl;j=1;

while(r!=NULL && j++n ;

if (r==NULL) printf("\n немає вузла %d ",i);

else printf("\n елемент %d дорівнює %f ",i,r->val);

2) печатка обох сусідів вузла(елемента), обумовленого покажчиком p (див. мал.4)



**Рис.4. Схема вибору сусідніх елементів.**

if((r=p->n)==NULL) printf("\n немає сусіда праворуч");

else printf("\n сусід праворуч %f", r->val);

if(dl==p) printf("\n немає сусіда ліворуч" );

else { r=dl;

while( r->n!=p ) r=r->n;

printf("\n лівий сусід %f", r->val);

}

3) видалення елемента, що випливає за вузлом, на який указує р (див. мал.5)



**Рис.5. Схема видалення елемента зі списку.**

if ((r=p->n)==NULL) printf("\n немає наступного");

p->n=r->n; free(r->n);

4) вставка нового вузла зі значенням new за елементом, визначеним покажчиком р (див. мал.6)



**Рис.6. Схема вставки елемента в список.**

r=malloc(1,sizeof(ND));

r->n=p->n; r->val=new; p->n=r;

5) часткове упорядкування списку в послідовність значень , s+t+1=l, так що K1'=K1; після упорядкування покажчик v указує на елемент K1' (див. мал.7)



**Рис.7. Схема часткового упорядкування списку.**

ND \*v;

float k1;

k1=dl->val;

r=dl;

while( r->n!=NULL )

{ v=r->n;

if (v->valn=v->n;

v->n=dl;

dl=v;

}

else r=v;

}

Кількість дій, необхідних для виконання зазначених операцій над списком у зв'язаному збереженні, оцінюється співвідношеннями: для операцій 1 і 2 - Q=l; для операцій 3 і 4 - Q=1; для операції 5 - Q=l.

#### *Організація двохзв‘язних списків*

Зв'язане збереження лінійного списку називається списком із двома зв'язками або двозв‘язним списком, якщо кожен елемент збереження має два компоненти покажчика (посилання на попередній і наступний елементи лінійного списку).

У програмі двозв‘язний список можна реалізувати за допомогою описів:

typedef struct ndd

{ float val; /\* значення елемента \*/

struct ndd \* n; /\* покажчик на наступний елемент \*/

struct ndd \* m; /\* покажчик на попередній елемент \*/

} NDD;

NDD \* dl, \* p, \* r;

Графічна інтерпретація методу зв'язаного збереження списку F=<2,5,7,1> як списку з двома зв'язками приведена на мал.8.



**Рис.8. Схема збереження двузв‘язного списку.**

Вставка нового вузла зі значенням new за елементом, обумовленим покажчиком p, здійснюється за допомогою операторів:

r=malloc(NDD);

r->val=new;

r->n=p->n;

(p->n)->m=r;

p->=r;

Видалення елемента, що випливає за вузлом, на який указує p

p->n=r;

p->n=(p->n)->n;

( (p->n)->n )->m=p;

free(r);

Зв'язане збереження лінійного списку називається циклічним списком, якщо його останній указує на перший елемент, а покажчик dl - на останній елемент списку.

Схема циклічного збереження списку F=<2,5,7,1> приведена на мал.9.



**Рис.9. Схема циклічного збереження списку.**

При рішенні конкретних задач можуть виникати різні види зв'язаного збереження.

Нехай на вході задана послідовність цілих чисел B1,B2,...,Bn з інтервалу від 1 до 9999, і нехай Fi (1<I по зростанню. Скласти процедуру для формування Fn у зв'язаному збереженні і повернення покажчика на його початок.

При рішенні задачі в кожен момент часу маємо упорядкований список Fi і при введенні елемента Bi+1 уставляємо його в потрібне місце списку Fi, одержуючи упорядкований список Fi+1. Тут можливі три варіанти: у списку немає елементів; число вставляється в початок списку; число вставляється в кінець списку. Щоб уніфікувати всі можливі варіанти, початковий список організуємо як зв'язаний список із двох елементів <0,1000>.

Розглянемо програму рішення поставленої задачі, у якій покажчики dl, r, p, v мають наступне значення: dl указує початок списку; p, v - два сусідніх вузли; r фіксує вузол, що містить чергове введене значення in.

#include

#include

typedef struct str1

{ float val;

struct str1 \*n; } ND;

main()

{ ND \*arrange(void);

ND \*p;

p=arrange();

while(p!=NULL)

{

printf("\n %f ",p->val);

p=p->n;

}

}

ND \*arrange() /\* формування упорядкованого списку \*/

{ ND \*dl, \*r, \*p, \*v;

float in=1;

char \*is;

dl=malloc(sizeof(ND));

dl->val=0; /\* перший елемент \*/

dl->n=r=malloc(sizeof(ND));

r->val=10000; r->n=NULL; /\* останній елемент \*/

while(1)

{

scanf(" %s",is);

if(\* is=='q') break;

in=atof(is);

r=malloc(sizeof(ND));

r->val=in;

p=dl;

v=p->n;

while(v->valn;

}

r->n=v;

p->n=r;

}

return(dl);

}

#### Стеки і черги

У залежності від методу доступу до елементів лінійного списку розрізняють різновиду лінійних списків називані стеком, чергою і двосторонньою чергою.

Стек - це кінцева послідовність деяких однотипних елементів - скалярних перемінних, масивів, структур або об'єднань, серед яких можуть бути й однакові. Стік позначається у виді: S= і представляє динамічну структуру даних; її кількість елементів заздалегідь не вказується й у процесі роботи, як правило змінюється. Якщо в стеці елементів ні, то він називається порожнім і позначається S=<>.

Припустимими операціями над стеком є:

- перевірка стека на порожнечу S=<>,

- додавання нового елемента Sn+1 у кінець стека - перетворення < S1,...,Sn> у < S1,...,Sn+1>;

- вилучення останнього елемента зі стека - перетворення < S1,...,Sn-1,Sn> у < S1,...,Sn-1>;

- доступ до його останнього елемента Sn, якщо стік не порожній.

Таким чином, операції додавання і видалення елемента, а також доступу до елемента виконуються тільки наприкінці списку. Стік можна представити як стопку книг на столі, де додавання або узяття нової книги можливо тільки зверху.

Черга - це лінійний список, де елементи віддаляються з початку списку, а додаються наприкінці списку (як звичайна черга в магазині).

Двостороння черга - це лінійний список, у якого операції додавання і видалення елементів і доступу до елементів можливі як спочатку так і наприкінці списку. Таку чергу можна представити як послідовність книг на полку, так що доступ до них можливий з обох кінців.

Реалізація стеков і черг у програмі може бути виконана у виді послідовного або зв'язаного збереження. Розглянемо приклади організації стека цими способами.

Однієї з форм представлення виражень є польський інверсний запис, що задає вираження так, що операції в ньому записуються в порядку виконання, а операнди знаходяться безпосередньо перед операцією.

Наприклад, вираз

(6+8)\*5-6/2

у польському інверсному записі має вигляд

6 8 + 5 \* 6 2 / -

Особливість такого запису полягає в тому, що значення вираження можна обчислити за один перегляд запису ліворуч праворуч, використовуючи стек, що до цього повинний бути порожній. Кожне нове число заноситься в стек, а операції виконуються над верхніми елементами стека, заміняючи ці елементи результатом операції. Для приведеного вираження динаміка зміни стека буде мати вигляд

S = <>; <6>; <6,8>; <14>; <14,5>; <70>;

<70,6>; <70,6,2>; <70,3>; <67>.

Нижче приведена функція eval, що обчислює значення вираження, заданого в масиві m у формі польського інверсного запису, причому m[i]>0 означає ненегативне число, а значення m[i]<0 операції. Як кодування операцій додавання, вирахування, множення і розподіли обрані негативні числа 1, 2, 3, 4. Для організації послідовного збереження стека використовується внутрішній масив stack. Параметрами функції є вхідний масив a і його довжина l.

float eval (float \*m, int l) { int p,n,i; float stack[50],c;

for(i=0; i < l ;i++) if ((n=m[i])<0) { c="st[p--];" switch(n) { case 1: stack[p]+="c;" break; case 2: stack[p]-="c;" break; case 3: stack[p]\*="c;" break; case 4: stack[p]/="c;" } } else stack[++p]="n;" return(stack[p]); }

Розглянемо іншу задачу. Нехай потрібно ввести деяку послідовність символів, що закінчується крапкою, і надрукувати неї в зворотному порядку (тобто якщо на вході буде "ABcEr-1." те на виході повинне бути "1-rEcBA"). Представлена нижче програма спочатку уводить усі символи послідовності, записуючи них у стек, а потім уміст стека друкується в зворотному порядку. Це основна особливість стека - чим пізніше елемент занесений у стек, тим раніш він буде витягнутий зі стека. Реалізація стека виконана в зв'язаному збереженні за допомогою покажчиків p і q на тип, іменований ім'ям STACK.

#include

typedef struct st /\* оголошення типу STACK \*/

{ char ch;

struct st \*ps; } STACK;

main()

{ STACK \*p,\*q;

char a;

p=NULL;

do /\* заповнення стека \*/

{ a=getch();

q=malloc(sizeof(STR1));

q->ps=p; p=q;

q->ch=a;

} while(a!='.');

do /\* печатка стека \*/

{ p=q->ps;free(q);q=p;

printf("%c",p->ch);

} while(p->ps!=NULL);

}

#### Стиснуте й індексне збереження лінійних списків

При збереженні великих обсягів інформації у формі лінійних списків небажано зберігати елементи з однаковим значенням, тому використовують різні методи стиску списків.

Стиснуте збереження. Нехай у списку B= кілька елементів мають однакове значення V, а список В'= виходить з B заміною кожного елемента Ki на пари Ki'=(і,Ki). Нехай далі B"= - підсписок В', що виходить викреслюванням усіх пар Ki=(і,V). Стиснутим збереженням У є метод збереження В", у якому елементи зі значенням V. Розрізняють послідовне стиснуте збереження і зв'язане стиснуте збереження. Наприклад, для списку B=, що містить кілька вузлів зі значенням Х, послідовного стиснутого і зв'язане стиснуте збереження, з умовчуванням елементів зі значенням Х, представлені на мал.22,23.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1,C | 3,Y | 6,S | 7,H | 9,T |
| **Рис.10. Послідовне стиснуте збереження списку.** | | | | |



**Рис.11. Зв'язне стиснуте збереження списку.**

Достоїнство стиснутого збереження списку при великому числі елементів зі значенням V полягає в можливості зменшення обсягу пам'яті для його збереження.

Пошук i-го елемента в зв'язаному стиснутому збереженні здійснюється методом повного перегляду, при послідовному збереженні - методом бінарного пошуку.

Переваги і недоліки послідовного стиснутого і зв'язаного стиснутого аналогічні перевагам і недолікам послідовного і зв'язаного збереження.

Розглянемо наступну задачу. На вході задані дві послідовності цілих чисел M=, N=, причому 92% елементів послідовності М дорівнюють нулеві. Скласти програму для обчислення суми добутків Mi \* Ni, і=1,2,...,10000.

Припустимо, що список М зберігається послідовно стисло в масиві структур m з оголошенням:

struct

{ int nm;

float val; } m[10000];

Для визначення кінця списку додамо ще один елемент із порядковим номером m[j].nm=10001, що називається стопером (stopper) і розташовується за останнім елементом стиснутого збереження списку в масиві m.

Програма для перебування шуканої суми має вигляд:

# include

main()

{ int і,j=0;

float inp,sum=0;

struct /\* оголошення масиву \*/

{ int nm; /\* структур \*/

float val; } m[10000];

for(i=0;i<10000;i++) /\* читання списку M \*/ { scanf("%f",&inp); if (inp!="0)" { m[j].nm="i;" m[j++].val="inp;" } } m[j].nm="10001;" /\* stopper \*/ for(i="0,j=0;" i<10000; i++) { scanf("%f",&inp); /\* читання списку N \*/ if(i="=m[j].nm)" /\* обчислення суми \*/ sum+="m[j++].val\*inp;" } printf( "сума добутків Mi\*Ni дорівнює %f",sum); }

Індексне збереження використовується для зменшення часу пошуку потрібного елемента в списку і полягає в наступному. Вихідний список B = розбивається на трохи підсписків У1,У2, ...,Вм таким чином, що кожен елемент списку В попадає тільки в один з підсписків, і додатково використовується індексний список з М елементами, що вказують на початок списків У1,У2, ...,Ум.

Вважається, що список зберігається індексно за допомогою підсписків B1,B2, ...,Bm і індексного списку X = , де ADGj - адреса початку підсписка Bj, j=1,M.

При індексному збереженні елемент До підсписка Bj має індекс j. Для одержання індексного збереження вихідний список У часто перетвориться в список В' шляхом включення в кожен вузол ще і його порядкового номера у вихідному списку В, а в j-ий елемент індексного списку Х, крім ADGj, може включатися деяка додаткова інформація про підсписок Bj. Розбивка списку В на підсписки здійснюється так, щоб всі елементи В, що володіють визначеною властивістю Рj, попадали в один підсписок Bj.

Достоїнством індексного збереження є те, що для перебування елемента К с заданою властивістю Pj досить переглянути тільки елементи підсписка Bj; його початок знаходиться по індексному списку Х, тому що для кожного ДО, що належить Bi, при і не рівному j властивість Pj не виконується.

У розбивці В часто використовується індексна функція G(K), що обчислює по елементі До його індекс j, тобто G(K)=j. Функція G звичайно залежить від позиції ДО, що позначається поз.K, у підсписку В або від значення визначеної частини компоненти ДО - її ключа.

Розглянемо список B= з елементами

ДО1=(17,Y), K2=(23,H), K3=(60,I), K4=(90,S), K5=(66,T),

K6=(77,T), K7=(50,U), K8=(88,W), K9=(30,S).

Якщо для розбивки цього списку на підсписки як індексну функцію взяти Ga(K)=1+(поз.K-1)/3, то список розділиться на три підсписка:

B1a=<(17,Y),(23,H),(60,I)>,

B2a=<(90,S),(66,T),(77,T)>,

B3a=<(50,U),(88,W),(30,S)>.

Додаючи усюди ще і початкову позицію елемента в списку, одержуємо:

B1a'=<(1,17,Y),(2,23,H),(3,60,I)>,

B2a'=<(4,90,S),(5,66,T),(6,77,T)>,

B3а'=<(7,50,U),(8,88,W),(9,30,S)>.

Якщо як індексну функцію вибрати іншу функцію Gb(K)=1+(поз.K-1)%3, то одержимо списки:

B1b"=<(1,17,Y),(4,90,S),(7,50,U)>,

B2b"=<(2,23,H),(5,66,T),(8,88,U)>,

B3b"=<(3,60,I),(6,77,T),(9,30,S)>.

Тепер для перебування вузла K6 досить переглянути тільки одну з трьох послідовностей (списків). При використанні функції Ga(K) це список B2а', а при функції Gb(K) список B3b".

Для індексної функції Gc(K)=1+K1/100, де K1 - перший компонент елемента ДО, знаходимо:

B1=<(17,Y),(23,H),(60,I),(90,S)>,

B2=<(66,T),(77,T)>,

B3=<(50,U),(88,W)>,

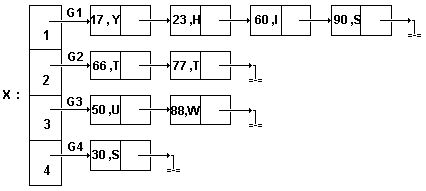
B4=<(30,S)>.

Щоб знайти тут вузол К с першим компонентом-ключем ДО1=77, досить переглянути список B2.

При реалізації індексного збереження застосовується методика А для збереження індексного списку Х (функція Ga(X) ) і методика C для збереження підсписків B1,B2,...,Bm (функція Gc(Bi)), тобто використовується, так називане, A-C індексне збереження.

У практиці часто використовується послідовно-зв'язане індексне збереження. Тому що звичайно довжина списку індексів відома, те його зручно зберігати послідовно, забезпечуючи прямій доступ до будь-якого елемента списку індексів. Підсписки B1,B2,...,Bm зберігаються пов'язано, що спрощує вставку і видалення вузлів(елементів). Зокрема, подібний метод збереження використовується в ЄС ЕОМ для організації, так званих, індексно-послідовних наборів даних, у яких доступ до окремих записів можливий як послідовно, так і за допомогою ключа.

Послідовно-зв‘язане індексне збереження для приведеного приклада зображене на мал.24, де X=.



**Рис.12.**

Розглянемо ще одну задачу. На вході задана послідовність цілих позитивних чисел, що закінчується нулем. Скласти процедуру для введення цієї послідовності й організації її індексного збереження таким чином, щоб числа, що збігаються в двох останніх цифрах, містилися в один підсписок.

Виберемо як індексну функцію G(K)=K%100+1, а як індексний список Х - масив з 100 елементів. Наступна функція вирішує поставлену задачу:

#include

#include

typedef struct nd

{ float val;

struct nd \*n; } ND;

int index (ND \*x[100])

{ ND \*p;

int i,j=0;

float inp;

for (i=0; i<100; i++) x[i]="NULL;" scanf("%d",&inp); while (inp!="0)" { j++; p="malloc(sizeof(ND));" i="inp%100+1;" p->val=inp;

p->n=x[i];

x[i]=p;

scanf("%d",&inp);

}

return j;

}

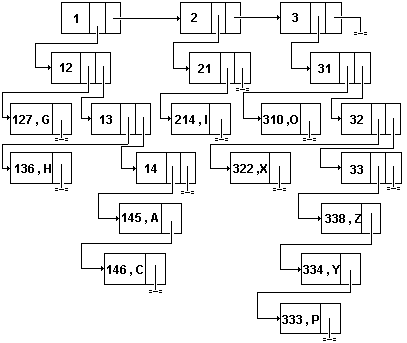
Значенням функції, що повертається, index буде число оброблених елементів списку.

Для індексного списку також може використовуватися індексне збереження. Нехай, наприклад, мається список B= з елементами

K1=(338,Z), K2=(145,A), K3=(136,H), K4=(214,I), K5 =(146,C),

K6=(334,Y), K7=(333,P), K8=(127,G), K9=(310,O), K10=(322,X).

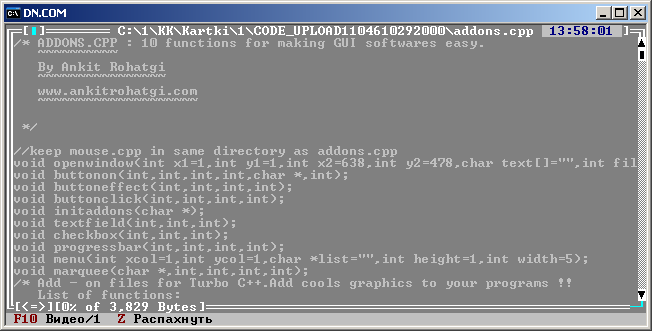
Потрібно розділити його на сімох підсписків, тобто X= таким чином, щоб у кожен список B1,B2,...,B7 попадали елементи, що збігаються в першому компоненті першими двома цифрами. Список Х, у свою чергу, будемо індексувати списком індексів Y=, щоб у кожен список Y1,Y2,Y3 попадали елементи з X, у яких у першому компоненті збігаються перші цифри. Якщо списки B1,B2,...,B7 зберігати пов'язано, а списки індексів X,Y індексно, те такий спосіб збереження списку B називається зв'язаним індексним збереженням. Графічне зображення цього збереження приведене на мал.25.



**Рис.13. зв'язане індексне збереження списку.**

**Практична частина**

Результатом нашої роботи є програмний комплекс для роботи (розробки) візитних карток (мал. 1). Для економії часу при розробці програми ми використали деякі вже готові функції для роботи з графікою.



Мал. 1. Лістинг модуля функцій для роботи з графікою.

Основною частиною програми є модуль для роботи з графічним форматом BMP (лістинг 1).

*Лістинг 1.*

/\* windows bmp format image storage \*/

BMPsave(int x,int y,int x1,int y1,char a[])

{

struct bmp

{unsigned char name[3];

unsigned char dummy[15];

unsigned int x;

unsigned char dummy1[2];

unsigned int y;

unsigned char dummy2[1];

}bmphead;

FILE \*fp,\*fq;

int xlen,ylen,i,j,count;

unsigned char c,ch;

fp=fopen(a,"wb");

fq=fopen("electron.bmp","rb");

/\*fq=fopen("electron.bmp","rb");\*/

if (fp==NULL) return(-1);

fread(&bmphead,sizeof(struct bmp),1,fq);

xlen=x1-x+1;

ylen=y1-y+1;

bmphead.x=xlen;

bmphead.y=ylen;

fwrite(&bmphead,sizeof(struct bmp),1,fp);

for(i=0;i<=1052;i++)

{

fread(&ch,1,1,fq);

fwrite(&ch,1,1,fp);

}

fclose(fq);

/\*hidemouse();\*/

count=0;

for(j=y1;j>=y;j--)

{

/\*setcolor(BLUE);

line(201+count,426,201+count,444);\*/

count++;

for(i=x;i<=x1;i++)

{c=getpixel(i,j);

switch(c)

{

case 0: c=0;break;

case 1: c=12;break;

case 2: c=4;break;

case 3: c=5;break;

case 4: c=16;break;

case 5: c=7;break;

case 6: c=13;break;

case 7: c=15;break;

case 8:c = 1;break;

case 9:c=10;break;

case 10: c=18;break;

case 11: c=24;break;

case 12: c=2;break;

case 13: c=21;break;

case 14: c=22;break;

case 15: c=14;break;

}

fwrite(&c,1,1,fp);

}

fwrite(&c,1,1,fp);

c=0;

fwrite(&c,1,1,fp);

}

count--;

/\*setcolor(LIGHTGRAY);

for(i=426;i<=444;i++)

line(201,i,201+count,i);\*/

/\*showmouse();\*/

fclose(fp);

return(0);

}

/\* windows bmp format image retrieval \*/

BMPload(int x,int y,char a[])

{

struct bmp

{unsigned char name[3];

unsigned char dummy[15];

unsigned int x;

unsigned char dummy1[2];

unsigned int y;

unsigned char dummy2[1];

}bmphead;

FILE \*fp,\*fq;

int xlen,ylen,i,j,diff,count;

unsigned char c,ch;

fp=fopen(a,"rb");

if (fp==NULL) return(-1);

fread(&bmphead,sizeof(struct bmp),1,fp);

if (!(bmphead.name[0]=='B' && bmphead.name[1]=='M'))

{fclose(fp);

return(-2);

}

if (bmphead.name[2]!=250 && bmphead.name[2]!=70)

{fclose(fp);

return(-3);

}

if (bmphead.x>600 || bmphead.y>400)

{fclose(fp);

return(-4);

}

for(i=0;i<=1052;i++)

{

fread(&ch,1,1,fp);

}

/\*hidemouse();\*/

if (bmphead.name[2]==250) diff=1;

else

if (bmphead.name[2]==70) diff=0;

count=0;

/\*hidemouse();\*/

/\*setcolor(WHITE);

for(i=82;i<=getmaxx()-12;i++)

line(i,47,i,getmaxy()-72); \*/

for(j=y+bmphead.y-2;j>=y;j--)

{

/\*setcolor(BLUE);

line(201+count,426,201+count,444);\*/

count++;

for(i=x;i<x+bmphead.x+diff;i++)

{

fread(&c,1,1,fp);

switch(c)

{

case 0: c=0;break;

case 12: c=1;break;

case 4: c=2;break;

case 5: c=3;break;

case 16: c=4;break;

case 7: c=5;break;

case 13: c=6;break;

case 15: c=7;break;

case 1:c = 8;break;

case 10:c=9;break;

case 18: c=10;break;

case 24: c=11;break;

case 2: c=12;break;

case 21: c=13;break;

case 22: c=14;break;

case 14: c=15;break;

}

putpixel(i-100,j-50,c);

}

if (bmphead.name[2]==250)

fread(&c,1,1,fp);

}

count--;

/\*setcolor(LIGHTGRAY);

for(i=426;i<=444;i++)

line(201,i,201+count,i);\*/

/\* showmouse();\*/

fclose(fp);

return(0);

}

Інтерфейс програми базується на роботі з мишкою та програмним інтерфейсом користувача (Лістинг 2).

*Лістинг 2.*

class mouse

{

private:

union REGS i,o;

struct SREGS s;

public:

mouse()

{

initmouse();

showmouseptr();

}

void initmouse()

{

i.x.ax=0;

int86(0x33,&i,&o);

}

void showmouseptr()

{

i.x.ax=1;

int86(0x33,&i,&o);

}

void hidemouseptr()

{

i.x.ax=2;

int86(0x33,&i,&o);

}

void getmousepos(int& button,int& x,int& y)

{

i.x.ax=3;

int86(0x33,&i,&o);

button= o.x.bx;

x=o.x.cx;

y=o.x.dx;

}

void restrictmouseptr(int x1,int y1,int x2,int y2)

{

i.x.ax=7;

i.x.cx=x1;

i.x.dx=x2;

int86(0x33,&i,&o);

i.x.ax=8;

i.x.cx=y1;

i.x.dx=y2;

int86(0x33,&i,&o);

}

void changemouseptr(int mark[50],int xstart,int ystart)

{

i.x.ax=9;

i.x.bx=xstart;

i.x.cx=ystart;

i.x.dx=(int)mark;

segread(&s);

s.es=s.ds;

int86x(0x33,&i,&i,&s);

}

};

Головний модуль програми відповідає за управління всіма цими компонентами (лістинг 3).

*Лістинг 3.*

/\* FILE NO: A

auxillary file for img3.cpp ie- Imager 3.1

\*/

void toolset(int);

void lift(int);

int imagearea();

void fillwindow();

void back(int,int,int,int,int);

void backtotal(int,int,int,int,int);

void myrect(int,int,int,int);

void ba(int,int,int,int,int);

void invertcolors();

void fliphorizontal();

void flipvertical();

void aboutwindow();

void textgetter(char \*);

void saveimage(int,int,int,int,char \*);

void loadimage(int,int,char \*);

void cut();

void copy();

void paste();

void changer(int);

void say(char \*);

void filemenu();

void editmenu();

void start()

{ m.hidemouseptr();

openwindow(0,0,639,479,title,1,1,1,3,9);

buttonon(5,20,35,35,"File",4);

buttonon(40,20,70,35,"Edit",4);

buttonon(75,20,105,35,"View",4);

buttonon(110,20,145,35,"About",3);

setfillstyle(1,15);

// defining buttons \*!\*

ba(0,0,0,0,0);

ba(5,20,35,35,1);

ba(40,20,70,35,2);

ba(75,20,105,35,3);

ba(110,20,145,35,4);

bar(100,50,635,450);

char \*labels[]={"Paint","Draw","Line","ellipse","Box","Text","Spray","Inv.Col","Cut","Copy","Paste","Select","Erase","H.flip","V.flip","Font","Filler","S.Save","Change","BMP","",""};

int cnta=0;

int tools=5;

for(int cnt=50;cnt<=300;cnt+=20)

{ buttonon(5,cnt,45,cnt+15,labels[cnta],3);

ba(5,cnt,45,cnt+15,tools);

tools++;

buttonon(50,cnt,95,cnt+15,labels[cnta+1],3);

ba(50,cnt,95,cnt+15,tools);tools++;

if(cnta<20)

{cnta+=2;}

}

setfillstyle(1,4);

bar(5,320,85,370);

buttonclick(5,320,85,370);

settextstyle(4,0,2);outtextxy(11,330,"Imager");settextstyle(2,0,4);

int cl=0;

for(int cntb=400;cntb<=410;cntb+=10)

{ for(int cntc=5;cntc<=80;cntc+=10)

{ setfillstyle(1,cl);

bar(cntc,cntb,cntc+10,cntb+10);

if(cl<=15)

{ cl++;}

}

}

setfillstyle(1,0);

bar(5,430,85,450);

/\*buttonon(25,455,70,470,"Modify",5);\*/

ba(639-13,3,639-3,12,31);

ba(5,400,85,420,32);

ba(100,50,635,450,33);

ofstream o("ccp.dat");

o<<0<<endl<<0;

o.close();

m.showmouseptr();

/\*music("imager3.snd");\*/

}

void ba(int x1,int y1,int x2,int y2,int n)

{ buttons[n][0]=x1;

buttons[n][1]=y1;

buttons[n][2]=x2;

buttons[n][3]=y2;

}

int imagearea()

{ m.getmousepos(button,x,y);

if((x>100)&&(y>50)&&(x<635)&&(y<450))

{return(1);}

else {return(0);}

}

void myrect(int x1,int y1,int x2,int y2)

{ line(x1,y1,x2,y1);

line(x1,y1,x1,y2);

line(x1,y2,x2,y2);

line(x2,y1,x2,y2);

}

void back(int x1,int y1,int x2,int y2,int mode)

{ int ch;char c2;

if(mode==0)

{

ofstream outfile("im~temp.dat");

for(int a=x1;a<=x2;a++)

{ for(int b=y1;b<=y2;b++)

{ ch=getpixel(a,b);

outfile<<(char)(ch+28);

}

}

outfile.close();

}

else if(mode==1)

{

ifstream infile("im~temp.dat");

for(int a=x1;a<=x2;a++)

{ for(int b=y1;b<=y2;b++)

{ infile.get(c2);ch=(int)c2;

putpixel(a,b,ch-28);

}

}

infile.close();

}

}

void fillwindow()

{ m.hidemouseptr();

back(200,150,350,200,0);

openwindow(200,150,350,200,"Filler",1,6,2,7,10);

buttonon(210,183,240,198,"ON",4);

ba(210,183,240,198,34);

buttonon(300,183,330,198,"OFF",4);

ba(300,183,330,198,35);

m.showmouseptr();

loop(34,35);

m.hidemouseptr();

back(200,150,350,200,1);

loopmain=1;

m.showmouseptr();

}

void fontwindow()

{ m.hidemouseptr();

back(200,100,405,350,0);

openwindow(200,100,405,350,"Choose Fonts",1,4,0,7,12);

settextstyle(2,HORIZ\_DIR,4);

for(int a=0;a<=11;a+=2)

{ if((a==9)||(a==10)) {settextstyle(3,HORIZ\_DIR,1);}

else{settextstyle(a,HORIZ\_DIR,1);}

buttonon(205,120+((a/2)\*30),300,120+25+((a/2)\*30),"Imager",2);

ba(205,120+((a/2)\*30),300,120+25+((a/2)\*30),37+a);

if((a+1==9)||(a+1==10)) {settextstyle(3,HORIZ\_DIR,1);}

else{settextstyle(a+1,HORIZ\_DIR,1);}

buttonon(305,120+((a/2)\*30),400,120+25+((a/2)\*30),"Imager",2);

ba(305,120+((a/2)\*30),400,120+25+((a/2)\*30),37+a+1);

}

int lb=0;char \*t[]={"1","2","3","4","5","6","7","8","9","10"};

settextstyle(2,0,4);

for(a=205;a<=355;a+=16)

{

/\*t='9'-(char)lb;

outtextxy(a+12,330,t);\*/

buttonon(a,330,a+15,345,t[lb],2);

ba(a,330,a+15,345,48+lb+1);

lb++;

}

m.showmouseptr();

loop(37,58);

loopmain=1;

fontwc=0;

m.hidemouseptr();

back(200,100,405,350,1);

m.showmouseptr();

}

void aboutwindow()

{ m.hidemouseptr();

back(200,100,450,200,0);

openwindow(200,100,450,200,title,1,4,0,7,0);

settextstyle(2,HORIZ\_DIR,4);

setcolor(14);

outtextxy(205,115,"Imager 3.1");

outtextxy(205,130," Made by Ankit Rohatgi");

outtextxy(205,145," www.ankitrohatgi.com");

m.showmouseptr();

m.getmousepos(button,x,y);

while(button==0)

{ m.getmousepos(button,x,y);

}

m.hidemouseptr();

back(200,100,450,200,1);

m.showmouseptr();

}

void toolset(int no)

{ m.hidemouseptr();

for(int cnt=50;cnt<=300;cnt+=20)

{ buttoneffect(5,cnt,45,cnt+15);

buttoneffect(50,cnt,95,cnt+15);

}

buttoneffect(5,20,35,35);

buttoneffect(40,20,70,35);

buttoneffect(75,20,105,35);

buttoneffect(110,20,145,35);

buttonclick(buttons[no][0],buttons[no][1],buttons[no][2],buttons[no][3]);

m.showmouseptr();

}

void lift(int no)

{ m.hidemouseptr();

buttoneffect(buttons[no][0],buttons[no][1],buttons[no][2],buttons[no][3]);

m.showmouseptr();

}

void invertcolors()

{ int cl;

for(int a=100;a<=635;a++)

{ for(int b=50;b<=450;b++)

{ cl=getpixel(a,b);

putpixel(a,b,15-cl);

}

}

}

void fliphorizontal()

{ int c1,c2;

for(int a=0;a<=267;a++)

{ for(int b=50;b<=450;b++)

{ c1=getpixel(101+a,b);

c2=getpixel(634-a,b);

putpixel(101+a,b,c2);

putpixel(634-a,b,c1);

}

}

}

void flipvertical()

{ int c1,c2;

for(int b=0;b<=200;b++)

{ for(int a=100;a<=635;a++)

{ c1=getpixel(a,50+b);

c2=getpixel(a,450-b);

putpixel(a,50+b,c2);

putpixel(a,450-b,c1);

}

}

}

void filemenu()

{ char \*filem[]={"New","Open","Save","Save As","Exit"};

m.hidemouseptr();

back(5,40,70,140,0);

menu(5,40,filem,5,7);

m.showmouseptr();

m.getmousepos(button,x,y);

int lc=1,mc=0;

while((x>5)&&(y>20)&&(x<60)&&(y<140)&&(lc==1))

{ m.getmousepos(button,x,y);

if(button==1)

{ if((y>40)&&(y<55)) { setfillstyle(1,15);bar(100,50,635,450);lc=0;

savename="tempo.dat";}

else if((y>55)&&(y<75))

{lc=0;mc=1;

m.hidemouseptr();

back(5,40,70,140,1);lift(1);

m.showmouseptr();

textgetter("Open-filename");

if(yotext[0]=='\0') {goto bop;}

loadimage(100,50,yotext);

/\*BMPload(100,50,yotext);\*/

savename=yotext;

bop:

}

else if((y>75)&&(y<90))

{ lc=0;mc=1;

m.hidemouseptr();

back(5,40,70,140,1);lift(1);

m.showmouseptr();

saveimage(100,50,635,450,savename);

/\*BMPsave(100,50,635,450,savename);\*/

}

else if((y>90)&&(y<105))

{ lc=0;mc=1;

m.hidemouseptr();

back(5,40,70,140,1);lift(1);

m.showmouseptr();

textgetter("Save As-filename");

if(yotext[0]=='\0') {goto bosa;}

saveimage(100,50,635,450,yotext);

/\*BMPsave(100,50,635,450,yotext);\*/

savename=yotext;

bosa:

}

else if((y>105)&&(y<120)) {closegraph();exit(1);}

}

}

if(mc==0) {

m.hidemouseptr();

back(5,40,70,140,1);lift(1);

m.showmouseptr();}

}

void editmenu()

{ char \*editm[]={"Cut","Copy","Paste"};

m.hidemouseptr();

back(40,35,85,90,0);

menu(40,35,editm,3,5);

m.showmouseptr();

m.getmousepos(button,x,y);

int lc=1;

delay(50);

while((x>40)&&(y>20)&&(x<85)&&(y<90)&&(lc==1))

{ m.getmousepos(button,x,y);

if(button==1)

{ if((y>40)&&(y<55)) {cut();lc=0;}

else if((y>55)&&(y<70)) {copy();lc=0;}

else if((y>70)&&(y<85)) {paste();lc=0;}

}

}

m.hidemouseptr();

back(40,35,85,90,1);

lift(2);

m.showmouseptr();

}

void cut()

{ setfillstyle(1,15);bar(select[0],select[1],select[2],select[3]);lift(13);

}

void copy()

{

backtotal(select[0],select[1],select[2],select[3],0);lift(14);

}

void paste()

{ backtotal(select[0],select[1],select[2],select[3],1);lift(15);

}

void backtotal(int x1,int y1,int x2,int y2,int mode)

{ int ch,width,height;char c2;

if(mode==0)

{

ofstream outfile("ccp.dat");

outfile<<x2-x1<<endl<<y2-y1<<endl;

for(int a=x1;a<=x2;a++)

{ for(int b=y1;b<=y2;b++)

{ ch=getpixel(a,b);

outfile<<(char)(ch+28);

}

}

outfile.close();

}

else if(mode==1)

{

setviewport(100,50,635,450,1);

ifstream infile("ccp.dat");

infile>>width>>height;

for(int a=0;a<=width;a++)

{ for(int b=0;b<=height;b++)

{ infile.get(c2);ch=(int)c2;

putpixel(a+x1-100,b+y1-50,ch-28);

}

}

infile.close();

setviewport(0,0,639,479,1);

}

}

void textgetter(char quest[])

{

m.hidemouseptr();

back(150,100,400,150,0);

openwindow(150,100,400,150,quest,1);

textfield(155,118,30);

m.showmouseptr();

setcolor(0);

settextstyle(2,0,4);

int xcn=0;int in2=0;

char in[2];

in[1]='\0'; /\*textcolor(4);gotoxy(23,9);gets(yotext);\*/

while(\*in!=(char)13)

{ st:

in[0]=getch();

if(in[0]==(char)13)

{ break;

}

else if(in[0]==(char)8) {in2-=6;xcn--;sound(900);nosound();setfillstyle(1,15);bar(158+in2,120,158+in2+textwidth("W"),120+textheight("Wg"));goto st;}

setfillstyle(1,15);bar(158+in2,120,158+in2+textwidth(in),120+textheight(in));

if(in2<=0) {in2=0;}

if(in2>=240) {in2=240;}

outtextxy(158+in2,120,in);

in2+=6;

yotext[xcn]=in[0];

xcn++;

}

yotext[xcn]='\0';

/\*cout<<yotext;\*/

m.hidemouseptr();

back(150,100,400,150,1);

m.showmouseptr();

}

void saveimage(int x1,int y1,int x2,int y2,char filename[])

{ int ch,width,height;char c2;

ofstream outfile(filename);

outfile<<x2-x1<<endl<<y2-y1<<endl;

for(int a=x1;a<=x2;a++)

{ for(int b=y1;b<=y2;b++)

{ ch=getpixel(a,b);

outfile<<(char)(ch+28);

}

}

outfile.close();

}

void loadimage(int x1,int y1,char filename[])

{

char c2;int width,height,ch;

ifstream infile(filename);

if(infile)

{

setviewport(100,50,635,450,1);

infile>>width>>height;

for(int a=0;a<=width;a++)

{ for(int b=0;b<=height;b++)

{ infile.get(c2);ch=(int)c2;

putpixel(a+x1-100,b+y1-50,ch-28);

}

}

setviewport(0,0,639,479,1);

}

else { say("file not found");}

infile.close();

}

void changer(int col)

{ for(int a=100;a<=635;a++)

{ for(int b=50;b<=450;b++)

{ if(col==getpixel(a,b))

{ putpixel(a,b,icolor);

}

}

}

}

void say(char text[])

{ m.hidemouseptr();

settextstyle(2,0,4);

int width=textwidth(text)+20;

int height=60;

back(150,100,150+width,100+height,0);

openwindow(150,100,150+width,100+height,title);

setcolor(0);

buttonon(160,100+height-17,180,100+height-2,"OK",3);

setcolor(0);

outtextxy(160,120,text);

m.showmouseptr();

int lc=1;

while(lc==1)

{ m.getmousepos(button,x,y);

if((x>160)&&(x<180)&&(y>100+height-17)&&(y<100+height-2)&&(button==1))

{ lc=0;

}

}

m.hidemouseptr();

back(150,100,150+width,100+height,1);

m.showmouseptr();

delay(90);

}

**Висновки**

В роботі були проаналізовані питання відносно теоретичної основи побудови програм з простим інтерфейсом користувача а також проблеми, які виникають при цьому. Результатом цього є розроблений програмний комплекс для роботи з візитними картками (розробка, та адміністрування). Зауважимо лише, що при цьому використовувалися готові функції для роботи з графікою.

Програма відрізняється від існуючих на ринку простотою інтерфейсу і настроювань.

**Література**

1. Касаткин А.И., Вальвачев А.Н. Профессиональное прогрпммирование на языке Си. Мн., 1992. 240 С.
2. Нейбауэр А. Моя первая программа на С/С++. П., 1995. 368 С.
3. Бруно Бабэ. Просто и ясно о Borland C++. М., 1996. 400 С.
4. Шамас Н.К. Основы С++ и обьектно-ориентированного программирования. К., 1996. 448 С.
5. Справочник по классам Borland C++ 4.0. К., 1994. 256 С.
6. ObjectWindows для C++. К., 1993., 208 С.
7. Том Сван. Программирование для Windows в Borland C++. М., 480 С.
8. Н. Барканати. Программирование игр для Windows на Borland C++. М., 1994. 512 С.