

Лекция 9. Многофайловые программы

Цели:

- освоить методику разработки многофайловых программ; методику написания алгоритмов с использованием функций, перевода таких алгоритмов на язык программирования C++ и разработки соответствующего проекта в среде Visual C++ 6.0.


При построении алгоритмов решения больших задач приходится строить основной и вспомогательные алгоритмы. Вспомогательные алгоритмы описывают решение подзадач основной задачи. Решение основной задачи сводится к последовательному решению всех подзадач.

При решении больших по объему прикладных задач на компьютере написание основной программы сводится к вызовам функций, которые позволяют решать подзадачи. Если описывать функции и основную программу в одном файле, то это затруднит её понимание при проверке. Эту проблему в языке C++ позволяет решить многофайловая программа.

Для реализации многофайловой программы в программный проект добавляют головные файлы, в которых описываются функции, затем при помощи директивы препроцессора `#include` эти файлы подключаются в основной программе, где описывается главная функция `main()`.

Реализация многофайловой программы:

Создать проект программы.

1. Выбрать пункт меню **Project** \diamond **Add to Project** \diamond **New** (для создания нового головного файла) и **Project** \diamond **Add to Project** \diamond **Files...** (для подключения уже созданного головного файла). Среда Visual C++ 6.0 будет выглядеть следующим образом (рис.10).
2. Появится диалоговое окно, в котором будет предложено выбрать тип подключаемого файла (**C/C++ HeaderFile**) и, если это нужно, набрать имя создаваемого головного файла в поле **File name**. После этого нужно нажать на кнопку  (рис.11).
3. Созданный головной файл будет находиться в той же папке, где и располагаются файлы проекта. В среде программирования файл можно найти на вкладке **FileView**, раскрыв папку **Header Files** (рис.12):
4. Открыв созданный головной файл, в окне редактора можно набрать текст функции. Для возможности использования созданной функции в своей программе необходимо в файле главной функции `<имя_проекта>.cpp` (на вкладке **FileView** окна рабочего пространства открыть папку **Source Files**) при помощи директивы `#include` подключить созданный головной файл (рис.13).

Пример 1. Дан одномерный массив целых чисел. Вычислить сумму максимального и первого элементов массива.

Ход выполнения работы

1. Основной алгоритм решения задачи (декомпозиция) выглядит следующим образом:
 1. ввод элементов массива;
 2. нахождение максимального элемента;
 3. вычисление суммы;
 4. вывод элементов массива на печать.

Каждая часть основного алгоритма является логически завершенной. Следовательно, для решения этой задачи нужно разработать три функции:

ввод элементов массива;

нахождение максимального элемента;

вывод элементов массива на печать.

Определим параметры и результаты работы каждой функции.

- Результатом работы первой функции должен быть сформированный массив. Этого можно достичь двумя способами: тип результата работы функции можно сделать указателем на тип элементов массива (1 вариант) или сделать формальным параметром указатель на тип элементов массива (2 вариант). Если в первом случае в теле функции будет необходимо использовать оператор `return` и будет возвращаться указатель на начало массива в ОП, то во втором случае этого не надо будет делать, поскольку в качестве фактического параметра будет передаваться адрес массива в ОП. При передаче указателя – фактического параметра в функцию произойдет заполнение ячеек ОП, которые будут отведены под массив. При этом их адреса не будут изменены. Еще одним параметром функции должна быть переменная, значение которой определяет количество вводимых переменных.
 - Результатом работы функции нахождения максимального элемента будет найденный элемент. Следовательно, в теле функции обязательно должен быть оператор `return`, который вернет найденное значение. Формальными параметрами функции будут указатель на начало массива и переменная, хранящая количество элементов массива.
 - Функция вывода элементов массива на печать не возвращает никакого результата, т.е. в теле функции оператор `return` отсутствует. Следовательно, формальными параметрами этой функции будут указатель на начало массива и переменная, хранящая количество элементов массива.
2. Создать проект (консольное приложение) с именем `massive`. В проекте создать и подключить два головных файла с именами `vvod.h` и `comp.h`. В первом файле описать функцию ввода и функцию вывода элементов одномерного массива, во втором – функцию, которая будет искать максимальный элемент.
 3. Записать тексты функций в головные файлы. Для этого открыть в окне рабочего пространства вкладку **FileView**, затем открыть папку **HeaderFiles**. В этой папке открыть файл `vvod.h` и записать в него тексты первых двух функций. В той же папке открыть файл `comp.h` и записать в него текст функции. Тексты программ, записанных в каждом файле:

файл `vvod.h`:

//функция ввода элементов массива: 1-ый вариант

`int*vv(inta[],intn)` //в качестве аргументов выступают одномерный

//массив и количество его элементов

{

`inti`;//локальная переменная – номер элемента массива. Область

//видимости переменной – тело функции

`for(i=0;i<=n-1;i++)`

```

{

cout<<"a["<<i<<"]="";

cin>>a[i];

}

return a; //возвращаем адрес на начало массива в ОП

}

//функция ввода элементов массива: 2-ой вариант

void vv(int *a, int n)

{

inti; //локальная переменная – номер элемента массива.

//Область видимости – тело функции

for(i=0; i<=n-1; i++)

{

cout<<"a"<<i<<"]="";

cin>>*(a+i);

}

}

//функция вывода элементов массива

void viv(int *a, int n)

{

inti; //локальная переменная – номер элемента массива.

//Область видимости – тело функции

for(i=0; i<=n-1; i++)

{

cout<<"a"<<i<<"]="<<*(a+i)<<" ";

}

cout<<endl;

```

```
}
```

файлcomp.h:

```
//функция поиска максимального элемента массива
```

```
int max(int *a, int n)
```

```
{
```

```
    inti;//локальная переменная – номер элемента массива.
```

```
//Область видимости – тело функции
```

```
    intmax_a;//локальная переменная – максимальный элемент массива.
```

```
//Область видимости – тело функции
```

```
    max_a=*a;
```

```
    for(i=1;i<=n-1;i++)
```

```
    {
```

```
        if(max_a<*(a+i))
```

```
        max_a=*(a+i);
```

```
    }
```

```
    return max_a;
```

```
}
```

4. Записать текст главной функции. Для этого открыть в окне рабочего пространства вкладку **FileView** и папку **Source Files**. В этой папке открыть файл massive.cpp и записать в него текст главной функции:

```
#include "stdio.h"
```

```
#include "stdlib.h"
```

```
#include "iostream.h"
```

```
#include "iomanip.h"
```

```
#include "vvod.h"
```

```
#include "comp.h"
```

```
int main()
```

```
{
```

```

intn, *p, s;

cout<<"Введите количество элементов массиваn="";

cin>>n;

p=(int*)malloc(n*sizeof(int));

//введем элементы массива при помощи нашей функции 2-ой

//вариант

vv(p,n);

// при втором варианте описания этой функции был бы вызов:

// p=vv(p,n);

//вычислим сумму

s=*p+max(p,n);

//выведем значение суммы на экран

cout<<"summa="<<s<<endl;

//выведем элементы массива

viv(p, n);

free(p);

return1;

}

```

Примечания:

- При написании программ с использованием функций и массивов нужно учитывать тот факт, что если массив (одномерный или двухмерный, целочисленный или символьный и т.д.) должен быть изменен в результате работы функции, то в качестве формального параметра выступает указатель на начало массива в ОП. Если же массив не изменяется в процессе выполнения функции, то в качестве формального параметра можно определить указатель на начало массива в ОП, но при этом надо следить за тем, чтобы элементы массива не изменялись в процессе выполнения операторов тела функции.
 - Как видно из примера имена фактических и формальных параметров не совпадают, что не влияет на правильность работы программы.
5. Откомпилировать файл massive.cpp. Запустить программу на выполнение.

Пример 2. Каждого студента можно описать при помощи характеристик: ФИО, курс, специальность, предмет1, предмет2, предмет3. Написать программу (с использованием файлов и функций), определяющую количество студентов:

1. сдавших сессию на «отлично»;
2. не сдавших хотя бы 1 экзамен.

Ход выполнения работы

1. Опишем основной алгоритм решения задачи (декомпозиция). Переменная var обозначает вариант работы:

var=-1

пока var!=0

ввод var

выбор var

случай 1: // создание и запись в файл

вызов функции создания и записи в файл

выход

случай 2: //открытие файла для добавления

вызов функции добавления данных в файл

выход

случай 3: // открытие файл в режиме чтения

//определение характеристик

вызов функции вычисления искомых величин

печать характеристик

выход

все_выбор

все_цикл

Определим входные параметры и результат работы каждой функции.

1. Функция создания и записи в файл. Функционально результат ее работы будет зависеть от имени файла, указателя на поток (файл) и режима открытия файла. Следовательно, эти величины и будут формальными параметрами этой функции. Данная функция будет возвращать результат – целое число 0 в случае неудачной попытки открытия файла и число 1, если функция отработала нормально.
2. Функция добавления данных в файл. Функционально результат ее работы будет зависеть от имени файла, указателя на поток (файл) и режима открытия файла. Следовательно, эти величины и будут формальными параметрами этой функции. Данная функция будет возвращать результат – целое число 0 в случае неудачной попытки открытия файла и число 1, если функция отработала нормально. Из п.п.а), б) следует, что можно написать одну функцию для реализации действий, описанных в этих двух пунктах.

3. Функция вычисления искомых величин. Функционально результат ее работы будет зависеть от имени файла, указателя на поток (файл) и режима открытия файла. Следовательно, эти величины и будут формальными параметрами этой функции. Данная функция будет возвращать результат – целое число 0 в случае неудачной попытки открытия файла и число 1, если функция отработала нормально. В список формальных параметров этой функции добавим еще два параметра: указатели на искомые величины (будем называть их характеристиками) – значения, расположенные по переданным адресам, будут изменены в процессе выполнения функции.

Программа должна быть универсальной с точки зрения пользователя. Для достижения этого воспользуемся оператором выбора switch(), в котором будут вызываться соответствующие функции. Работа с файлом осуществляется не напрямую, а через динамический массив структур. Для ввода-вывода значений будут использоваться потоковые функции.

2. Создать проект (консольное приложение) с именем student. В проекте создать и подключить один головной файл с именем my_function.h. Открыть этот файл в окне рабочего пространства и описать в нем все функции. Тексты функций:

файл my_function.h:

```
//описание структурного типа
```

```
typedef struct
```

```
{
```

```
char fio[30];
```

```
int kurs;
```

```
char spec[30];
```

```
int hist;
```

```
int math;
```

```
int phis;
```

```
} stud;
```

```
//функция записи или добавления данных в файл
```

```
int write(char *name, FILE *pf, char *rezh)
```

```
{
```

```
stud *st;
```

```
int i;
```

```
long int n;
```

```
cout<<"n=";
```

```
cin>>n;
```

```

st=(stud*)malloc(n*sizeof(stud));

//заполнение массива структур

for(i=0;i<=n-1;i++)

{

cout<<"fio=";

cin>>((st+i)->fio);

cout<<"kurs=";

cin>>((st+i)->kurs);

cout<<"spec=";

cin>>((st+i)->spec);

cout<<"history=";

cin>>((st+i)->hist);

cout<<"math=";

cin>>((st+i)->math);

cout<<"phis=";

cin>>((st+i)->phis);

}

//файл pfс именем папеоткрывается в режим rezh

if((pf=fopen(name,rezh))==NULL)

{

printf("файл не открыт\n");

return 0;

}

//запись в файл

fwrite(st, sizeof(stud), n, pf);

fclose(pf);

free(st);

```



```

return 1;

}

//функциявычисленияискомыхвеличин

int read(char *name, FILE *pf, char *rezh, int *c_5, int *c_2)

{

stud *st;

int i;

long int n;

//локальные переменные cnt5 и cnt2 нужны для вычисления

//характеристик

int cnt5,cnt2;

// файл pfc именемnameоткрывается в режимerezh

if((pf=fopen(name,rezh))==NULL)

{

printf("файл не открыт\n");return 0;

}

//определение длины файла в байтах

fseek(pf, 0,SEEK_END);

n=ftell(pf);

//определение количества записей в файле

n=n/sizeof(stud);

st=(stud*)malloc(n*sizeof(stud));

//возвращаем указатель в начало файла

rewind(pf);

//заполнение массива структур

fread(st,sizeof(stud),n,pf);

cnt5=cnt2=0;

```

```

for (i=0;i<=n-1;i++)

{ //определяем количество отличников

if((st+i)->hist==5&&(st+i)->math==5&&(st+i)->phis==5)

cnt5++;

//определяем количество студентов, не сдавших хотя бы один

//экзамен

if((st+i)->hist==2||(st+i)->math==2||(st+i)->phis==2)

cnt2++;

}

fclose(pf);

free(st);

//возвращение найденных характеристик в главную программу

//через указатели

*c_5=cnt5;

*c_2=cnt2;

return 1;

}

```

Примечания:

1. В списке формальных параметров функции read() присутствуют два параметра-указателя. Это сделано для того, чтобы можно было при помощи функции возвращать не одну, а две величины (характеристики). Таким образом, если в результате работы функции должно быть получено более одной результирующей величины, то одну из этих величин можно возвращать при помощи оператора return, а другие добавлять в список формальных параметров как указатели на искомые переменные. В этом случае напрямую использовать эти параметры в операторах тела функции (например, в выражениях) не рекомендуется. Целесообразнее использовать локальные переменные для вычисления характеристик (в нашем примере это cnt2 и cnt5). Затем, полученные значения этих переменных записать по адресам ОП, переданным в функцию через указатели – фактические параметры.
2. Очевидно, что с использованием функций текст главной функции main() значительно упростился и стал более понятен при чтении. Намного проще становится отладка программы, поскольку можно выявить неправильную работу всей программы, исследуя ее по частям (функциям).
4. Откомпилировать файл massive.cpp. Запустить программу на выполнение.