Динамическое выделение памяти

C++ поддерживает три основных типа выделения (или ещё «распределения») памяти:

1. Статическое выделение памяти выполняется для статических и глобальных переменных. Память выделяется один раз, при запуске программы, и сохраняется на протяжении работы всей программы. Статические переменные мы еще не проходили. Использование же глобальных переменных может привести неопытных программистов к серьезным, плохо контролируемым ошибкам, поэтому применять их в лабораторных работах вам запрещено.
2. Автоматическое выделение памяти выполняется для параметров функции и локальных переменных. Память выделяется при входе в блок, в котором находятся эти переменные, и удаляется при выходе из него.
3. Динамическое выделение памяти. Сегодня это является темой нашей лекции

Как статическое, так и автоматическое распределение памяти имеют два общих свойства:

- Размер переменной/массива должен быть известен во время компиляции.

- Выделение и освобождение памяти происходит автоматически (когда переменная создаётся/уничтожается).

Это бывает не очень удобно по нескольким причинам:

1. Мы не всегда точно знаем размеры массива, который нам будет необходим, поэтому выделяем память с запасом, которого может оказаться слишком много, а может и не хватить. В первом случае мы теряем память, а во втором может наступить переполнение массива, что приведет к сбою при выполнении программы.

2. Во-вторых, память для большинства обычных переменных (включая фиксированные массивы) выделяется из специального резервуара памяти — стека. Объём памяти стека в программе, как правило, невелик: в Visual Studio он по умолчанию равен 1МБ. Если вы превысите это значение, то произойдёт *переполнение стека*, и операционная система автоматически завершит выполнение вашей программы.

Эти проблемы легко устраняются с помощью динамического выделения памяти. Динамическое выделение памяти — это способ запроса памяти из операционной системы запущенными программами по надобности. Эта память

не выделяется из ограниченной памяти стека программы, а из гораздо большего хранилища, управляемого операционной системой — кучи. На современных компьютерах размер кучи может составлять гигабайты памяти.

Для динамического выделения памяти для одной переменной используется **оператор new:**

Для доступа к выделенной памяти создаётся указатель

Оператор new возвращает указатель, содержащий адрес выделенной памяти.

```
int *psub;  // указатель на начало массива

psub=new  int [n]; // динамическое выделение области памяти
```

Когда уже всё, что нужно было, выполнено с динамически выделенной переменной или массивом — нужно явно указать C++ освободить эту память. Для переменных это выполняется с помощью оператора **delete,** для массивов **delete []**

Оператор delete на самом деле ничего не удаляет. Он просто возвращает память, которая была выделена ранее, обратно в операционную систему. Затем операционная система может переназначить эту память другому приложению (или этому же снова).

Работа с файлами

В материале данной темы используется понятие классов.
Класс – термин объектно-ориентированного программирования, с которым мы познакомимся на следующей лекции

Файл данных – это файл, в котором находятся данные. Файл присутствует на каком-либо носителе. Он для обмена данными должен быть открыт, по заверении процесса – закрыт.

Для работы с файлами мы используем классы файловых потоков
stream – поток – относится к любому переносу данных от источника к приемнику (последовательность байтов).

Для использования файловых потоков необходимо подключить к программе заголовочный файл <fstream>.

Для обмена информацией с файлом приложение должно создать объект класса *ifstream* для чтения из файла, или объект класса ofstream для вывода в

файл. После этого файл должен быть открыт, а при завершении обмена – закрыт. Описание классов *ifstream и ofstream* находится в заголовочном файле *<fstream>,* отвечающим за потоковый ввод-вывод в файлы.

Потоковые классы предназначены для управления потоками данных между ОП (оперативной памятью) и внешними устройствами. Стандартные объекты cin(связь с клавиатурой) и cout(связь с экраном монитора) и операции ввода-вывода (>> ,<<) описаны в заголовочном файле *<iostream>.*

Потоки определяются последовательностью байтов и не зависят от конкретного устройства, с которым производится обмен. *Входной поток* – это данные, которые вводятся в ОП. *Выходной поток* – это данные, которые выводятся из ОП.

Итак, потоки:

Стандартные потоки - от клавиатуры и на зкран.
Файловые потоки – предназначены для обмена с файлами.

Для поддержания потоков в библиотеке С++ содержится целая иерархия классов. Мы будем пользоваться следующими классами:

- ifstream - класс входных потоков
- ofstream - класс выходных потоков

Пример работы с файлом :

```
// чтение из файла
  ifstream fin;
  fin.open(“myfile1.txt”);
  if(fin)
    fin>>number;
  fin.close();
//запись в файл
  std::ofstream fout;
  fout.open(“myfile2.txt”);
  fout<<”Number=”<<numer<<”\n”;
  fout.close();
```

Вывод: файловый ввод- вывод связан с классами ifstream и ofstream, для чего требуется подключить заголовочный файл fstream. Последовательность действий:

- Создание объекта- потока (fin,fout)
- Открытие его в заданном режиме (fin.open(),fout.open()).

- Выполнение ввода-вывода данных (fin>>…, fout<<…)
- Закрытие объекта- потока (fin.close(),fout.close()).

Приведем полный текст программы для лабораторной работы №1:

```
//Проект Lab1_struct


typedef struct  {
   char surname[10];
   int time;              // шаблон структуры
   float price;

}subscriber;
```

Замечание 1

Для описания шаблона можно использовать оператор typedef (задание нового имени типу). Ключевое слово typedef делает программу более ясной.

```
int input(subscriber *psub, int n);
      float goal(subscriber *psub, int n); //прототипы функций
int output(subscriber *psub, int n);



int input(subscriber *psub,int n) {
   ifstream fin;
   fin.open("myfile1.txt");
   if(fin){
     for(int i=0;i<n;i++)
       fin>>(psub+i)->surname>>(psub+i)->time>>(psub+i)->price;
     cout<<"OK-file is read"<<endl;
   }
   else cout<<"Check up the name of the file";
   fin.close();

   cout<<"         CHECK up the content of the file"<<endl<<endl;

   for(int i=0;i<n;i++)
      cout<<" "<<(psub+i)->surname<<" "<<(psub+i)->time<<" "<<(psub+i)->price<<endl;
   cout<<endl;
   return 0;
}
```

```
float goal(subscriber *psub,int n) {
    cout<<"edit:"<<endl;
    cout<<"input surname:"<<endl;
    cin>>(psub+1)->surname;
    cout<<"input time:"<<endl;
    cin>>(psub+1)->time;
    cout<<"input price:"<<endl;
    cin>>(psub+1)->price;
    float sumprice=0;
    for(int i=0;i<n;i++)
        sumprice+=(psub+i)->price*(psub+i)->time;

    return sumprice;
}


int output(subscriber *psub,int n) {
    std::ofstream fout;
    fout.open("myfile2.txt");
    for(int i=0;i<n;i++)
        fout<<(psub+i)->surname<<" "<<(psub+i)->time<<" "<<(psub+i)->price<<endl;
    fout.close();
    return 0;
}

int main() {
    int n,m;
    float sumprice;

    cout<<"input n=";
    cin>>n;
    subscriber *psub;  // указатель на начало массива структур
   psub=new  subscriber[n]; // динамическое выделение области
памяти



    do{
            cout<<endl;
            cout<<"    1-input and check";
            cout<<"    2-edit";
            cout<<"    3- output";
            cout<<"    4- exit";
            cout<<endl;
            cin>>m;
```

```
        switch(m){
           case 1:  input(psub,n);  break;
           case 2:  sumprice=goal(psub,n);
                    std::cout<<"sumprice="<<sumprice;break;
           case 3:  output(psub,n);
                    std::cout<<"information is saved";break;
           case 4:  m=0;break;
           }
     }while(m!=0);

    delete[] psub;
    psub=NULL;
    return 0;
}
```

Разработка программы закончена. Осталось откомпилировать и запустить программу

Замечание 2

Существует несколько способов доступа к отдельным полям массива структур

1) *psub[i]. surname*
2) *(*(psub+i)).surname*
3) *(psub+i)->surname*

В п.п. 1) и 2) мы обращаемся к ***содержимому ячейки*** *памяти по указному адресу, но из этого содержимого извлекаем только ту информацию, которая соответствует выбранному нами полю (surname), имя которого записывается после операции точка.*

В п. 3) показан доступ к нужному полю непосредственно ***через указатель****, для чего используется операция стрелочка.*

Замечание 3

В программе используется оператор выбора ***switch().***
switch() - *частный случай оператора if.*
Используется обычно для организации меню.
В нашем случае m - выражение селектор. Его значение вводится с клавиатуры. Если m ввели 1, тогда
case 1: вызов функции input()
если m ввели 2 обращение к функции goal(); и т. д.

В лабораторной работе №1 организация меню не обязательна