Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информационных систем и технологий

**Лабораторная работа №10(1)**

**«**Классы хранения и видимость переменных**»**

**Вариант 22**

|  |  |
| --- | --- |
| Выполнил: | студент группы 478105  Писарик Мирослав Игоревич |
| Проверил: | ассистент  кафедры ИЭФ Канаш Алексей Васильевич |

Минск 2024

**Теоретические сведения**

1. **Исходные файлы и объявление переменных**

Обычная СИ-программа представляет собой определение функции main, которая для выполнения необходимых действий вызывает другие функции. Приведенные в части 1 примеры программ представляли собой один исходный файл, содержащий все необходимые для выполнения программы функции. Связь между функциями осуществлялась по данным посредством передачи параметров и возврата значений функций. Но компилятор языка Си позволяет также разбить программу на несколько отдельных частей (исходных файлов), оттранслировать каждую часть отдельно и затем объединить все части в один выполняемый файл при помощи редактора связей. При такой структуре исходной программы функции, находящиеся в разных исходных файлах, могут использовать глобальные внешние переменные. Все функции в языке Си по определению внешние и всегда доступны из любых файлов. Например, если программа состоит из двух исходных файлов, как показано на рис. 9.1, то функция main может вызывать любую из трех функций fun1, fun2, fun3, а каждая из этих функций может вызывать любую другую.

|  |  |  |
| --- | --- | --- |
| main ()  { ...  }  fun1()  { ...  } | . | fun2()  { ...  }  fun3()  { ...  } |
| file1.c | File2.c |
| Рис. 9.1. Пример программы из двух файлов | | |

Для того чтобы определяемая функция могла выполнять какие-либо действия, она должна использовать переменные. В языке Си все переменные должны быть объявлены до их использования. Объявления устанавливают соответствие имени и атрибутов переменной, функции или типа. Определение переменной вызывает выделение памяти для хранения ее значения. Класс выделяемой памяти определяется спецификатором класса памяти и определяет время жизни и область видимости переменной, связанные с понятием блока программы. В языке Си блоком считается последовательность объявлений, определений и операторов, заключенная в фигурные скобки. Существуют два вида блоков – составной оператор и определение функции, состоящее из составного оператора, являющегося телом функции, и предшествующего телу заголовка функции (в который входят имя функции, типы возвращаемого значения и формальных параметров). Блоки могут включать в себя составные операторы, но не определения функций. Внутренний блок называется вложенным, а внешний блок – объемлющим.

**Время жизни** – это интервал времени выполнения программы, в течение которого программный объект (переменная или функция) существует. Время жизни переменной может быть локальным или глобальным. Переменная с глобальным временем жизни имеет распределенную для нее память и определенное значение на протяжении всего времени выполнения программы, начиная с момента выполнения объявления этой переменной. Переменная с локальным временем жизни имеет распределенную для него память и определенное значение только во время выполнения блока, в котором эта переменная определена или объявлена. При каждом входе в блок для локальной переменной распределяется новая память, которая освобождается при выходе из блока.

**Область видимости** – это часть текста программы, в которой может быть использован данный объект. Объект считается видимым в блоке или в исходном файле, если в этом блоке или файле известны имя и тип объекта. Объект может быть видимым в пределах блока, исходного файла или во всех исходных файлах, образующих программу. Это зависит от того, на каком уровне объявлен объект: на внутреннем, т.е. внутри некоторого блока, или на внешнем, т.е. вне всех блоков. Если объект объявлен внутри блока, то он видим в этом блоке, и во всех внутренних блоках. Если объект объявлен на внешнем уровне, то он видим от точки его объявления до конца данного исходного файла.

**Спецификатор класса памяти** в объявлении переменной может быть auto, register, static или extern. Если класс памяти не указан, то он определяется по умолчанию из контекста объявления. Объекты классов auto и register имеют локальное время жизни. Спецификаторы static и extern определяют объекты с глобальным временем жизни. При объявлении переменной на внутреннем уровне может быть использован любой из четырех спецификаторов класса памяти, а если он не указан, то подразумевается класс памяти auto. Переменная с классом памяти auto имеет локальное время жизни и видна только в блоке, в котором объявлена. Память для такой переменной выделяется при входе в блок и освобождается при выходе из блока. При повторном входе в блок этой переменной может быть выделен другой участок памяти. Переменная с классом памяти auto автоматически не инициализируется. Она должна быть проинициализирована явно при объявлении путем присвоения ей начального значения. Значение неинициализированной переменной с классом памяти auto считается неопределенным.

Спецификатор класса памяти register предписывает компилятору распределить память для переменной в регистре, если это представляется возможным. Использование регистровой памяти обычно приводит к сокращению времени доступа к переменной. Переменная, объявленная с классом памяти register, имеет ту же область видимости, что и переменная auto. Число регистров, которые можно использовать для значений переменных, ограничено возможностями компьютера, и в том случае, если компилятор не имеет в распоряжении свободных регистров, то переменной выделяется память как для класса auto. Класс памяти register может быть указан только для переменных с типом int или указателей с размером равным размеру int.

Переменные, объявленные на внутреннем уровне со спецификатором класса памяти static, обеспечивают возможность сохранить значение переменной при выходе из блока и использовать его при повторном входе в блок. Такая переменная имеет глобальное время жизни и область видимости внутри блока, в котором она объявлена. В отличие от переменных с классом auto, память для которых выделяется в стеке, для переменных с классом static память выделяется в сегменте данных, и поэтому их значение сохраняется при выходе из блока.

В приведенном примере объявлены три разные переменные с классом памяти static, имеющие одинаковые имена i. Каждая из этих переменных имеет глобальное время жизни, но видима только в том блоке (функции), в которой она объявлена. Эти переменные можно использовать для подсчета числа обращений к каждой из трех функций.

Переменные класса памяти static могут быть инициализированы константным выражением. Если явной инициализации нет, то такой переменной присваивается нулевое значение. При инициализации константным адресным выражением можно использовать адреса любых внешних объектов, кроме адресов объектов с классом памяти auto, так как адрес последних не является константой и изменяется при каждом входе в блок. Инициализация выполняется один раз при первом входе в блок.

Переменная, объявленная локально с классом памяти extern, является ссылкой на переменную с тем же самым именем, определенную глобально в одном из исходных файлов программы. Цель такого объявления состоит в том, чтобы сделать определение переменной глобального уровня видимым внутри блока.

Объявление переменной i[] как extern в приведенном примере делает ее видимой внутри функции fun1. Определение этой переменной находится в файле file2.c на глобальном уровне и должно быть только одно, в то время как объявлений с классом памяти extern может быть несколько.

Объявление с классом памяти extern требует при необходимости использовать переменную, описанную в текущем исходном файле, но ниже по тексту программы, т.е. до выполнения ее глобального определения. Следующий пример иллюстрирует такое использование переменной с именем st.

Объявление переменной со спецификатором extern информирует компилятор о том, что память для переменной выделять не требуется, так как это выполнено где-то в другом месте программы.

При объявлении переменных на глобальном уровне может быть использован спецификатор класса памяти static или extern, а также можно объявлять переменные без указания класса памяти. Классы памяти auto и register для глобального объявления недопустимы. Объявление переменных на глобальном уровне – это или определение переменных, или ссылки на определения, сделанные в другом месте программы. Объявление глобальной переменной, которое инициализирует эту переменную (явно или неявно), является определением переменной. Определение на глобальном уровне может задаваться в следующих формах:

1. Переменная объявлена с классом памяти static. Такая переменная может быть инициализирована явно константным выражением, или по умолчанию нулевым значением. То есть обявления static int i=0 и static int i эквивалентны, и в обоих случаях переменной i будет присвоено значение 0.

2. Переменная объявлена без указания класса памяти, но с явной инициализацией. Такой переменной по умолчанию присваивается класс памяти static. То есть объявления int i=1 и static int i=1 будут эквивалентны.

Переменная, объявленная глобально, видима в пределах остатка исходного файла, в котором она определена. Выше своего описания и в других исходных файлах эта переменная невидима (если только она не объявлена с классом extern). Глобальная переменная может быть определена только один раз в пределах своей области видимости. В другом исходном файле может быть объявлена другая глобальная переменная с таким же именем и с классом памяти static, конфликта при этом не возникает, так как каждая из этих переменных будет видимой только в своем исходном файле. Спецификатор класса памяти extern для глобальных переменных используется, как и для локального объявления, в качестве ссылки на переменную, объявленную в другом месте программы, т.е. для расширения области видимости переменной. При таком объявлении область видимости переменной расширяется до конца исходного файла, в котором сделано объявление. В объявлениях с классом памяти extern не допускается инициализация, так как эти объявления ссылаются на уже существующие и определенные ранее переменные. Переменная, на которую делается ссылка с помощью спецификатора extern, может быть определена только один раз в одном из исходных файлов программы.

1. **Объявление функций**

Функции всегда определяются глобально. Они могут быть объявлены с классом памяти static или extern. Объявления функций на локальном и глобальном уровнях имеют одинаковый смысл.

Правила определения области видимости для функций отличаются от правил видимости для переменных и состоят в следующем:

1. Функция, объявленная как static, видима в пределах того файла, в котором она определена. Каждая функция может вызвать другую функцию с классом памяти static из своего исходного файла, но не может вызвать функцию определенную с классом static в другом исходном файле. Разные функции с классом памяти static, имеющие одинаковые имена, могут быть определены в разных исходных файлах, и это не ведет к конфликту.

2.  Функция, объявленная с классом памяти extern, видима в пределах всех исходных файлов программы. Любая функция может вызывать функции с классом памяти extern.

3.  Если в объявлении функции отсутствует спецификатор класса памяти, то по умолчанию принимается класс extern. Все объекты с классом памяти extern компилятор помещает в объектном файле в специальную таблицу внешних ссылок, которая используется редактором связей для разрешения внешних ссылок. Часть внешних ссылок порождается компилятором при обращениях к библиотечным функциям Си, поэтому для разрешения этих ссылок редактору связей должны быть доступны соответствующие библиотеки функций.

Приведем пример программы, текст которой размещен в двух файлах (для связи функций программы используется внешняя переменная i).

Индивидуальное задание

22. Разработать программу оценки продаж театральных билетов от времени года. Данные о продажах хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по величине прибыли.

menu.cpp:

#**include** <iostream>  
**using** **namespace** std;  
**extern** **int** userChoice;  
#**include** "components.cpp"  
   
**int** **main**() {  
    **bool** running = true;  
   
    **while** (running) {  
        cout << "Press 1 to enter information about tickets\n";  
        //TODO add checker of at least one run func of fetchingData till running changing  
        cout << "Press 2 to change information\n";  
        cout << "Press 3 to output information\n";  
        cout << "Press 0 to quit\n";  
        userChoiceValidation();  
   
        **switch** (userChoice) {  
            **case** 1:  
                fetchingData();  
                cout << "Tickets data successfully fetched\n";  
                **break**;  
            **case** 2:  
                changeTicketsData();  
                cout << "Tickets data successfully changed\n";  
                **break**;  
            **case** 3:  
                outputData();  
                **break**;  
            **case** 0:  
                running = false;  
                **break**;  
            **default**:  
                cout << "Enter correct menu option\n";  
                **break**;  
        }  
    }  
   
    **return** 0;  
}

components.cpp:

#**include** <algorithm>  
#**include** <iostream>  
#**include** <limits>  
**using** **namespace** std;  
  
**int** userChoice;  
**const** **int** SEASON\_ARR\_LENGTH = 4;  
  
**struct** **ticketsDataSeason** {  
    **int** soldTicketsAmount;  
    **int** price;  
    string yearSeason;  
    **int** ticketsRevenue = 0;  
};  
  
**static** ticketsDataSeason allTicketsDataSeason[SEASON\_ARR\_LENGTH];  
  
**void** **fetchingData**() {  
    **for** (**int** i = 0; i < SEASON\_ARR\_LENGTH; i++) {  
        cout << "Enter season: ";  
        cin >> allTicketsDataSeason[i].yearSeason;  
  
        **while** (true) {  
            cout << "Enter amount of sold tickets: ";  
            cin >> allTicketsDataSeason[i].soldTicketsAmount;  
            **if** (cin.fail()) {  
                cin.clear();  
                cin.ignore(numeric\_limits<streamsize>::max(), '\n');  
                cout << "Invalid input! Please enter a number.\n";  
            } **else** {  
                **break**;  
            }  
        }  
  
        **while** (true) {  
            cout << "Enter ticket price: ";  
            cin >> allTicketsDataSeason[i].price;  
            **if** (cin.fail()) {  
                cin.clear();  
                cin.ignore(numeric\_limits<streamsize>::max(), '\n');  
                cout << "Invalid input! Please enter a number.\n";  
            } **else** {  
                **break**;  
            }  
        }  
    }  
}  
  
**void** **revenueCalc**() {  
    **for** (**int** i = 0; i < SEASON\_ARR\_LENGTH; i++) {  
        allTicketsDataSeason[i].ticketsRevenue = allTicketsDataSeason[i].price \* allTicketsDataSeason[i].soldTicketsAmount;  
    }  
}  
  
**void** **changeTicketsData**() {  
    string searchingSeason;  
    cout << "Enter season name which information you want to change: ";  
    cin >> searchingSeason;  
  
    **for** (**int** i = 0; i < SEASON\_ARR\_LENGTH; i++) {  
        **if** (searchingSeason == allTicketsDataSeason[i].yearSeason) {  
            cout << "Press 1 to Change ticket price\n";  
            cout << "Press 2 to Change amount of sold tickets\n";  
            cout << "Press 0 to quit changing\n";  
            **int** p;  
            cin >> p;  
  
            **switch** (p) {  
                **case** 1:  
                    **while** (true) {  
                        cout << "Enter new ticket price: ";  
                        cin >> allTicketsDataSeason[i].price;  
                        **if** (cin.fail()) {  
                            cin.clear();  
                            cin.ignore(numeric\_limits<streamsize>::max(), '\n');  
                            cout << "Invalid input! Please enter a number.\n";  
                        } **else** {  
                            **break**;  
                        }  
                    }  
                    **break**;  
                **case** 2:  
                    **while** (true) {  
                        cout << "Enter new amount of sold tickets: ";  
                        cin >> allTicketsDataSeason[i].soldTicketsAmount;  
                        **if** (cin.fail()) {  
                            cin.clear();  
                            cin.ignore(numeric\_limits<streamsize>::max(), '\n');  
                            cout << "Invalid input! Please enter a number.\n";  
                        } **else** {  
                            **break**;  
                        }  
                    }  
                    **break**;  
                **case** 0:  
                    **break**;  
                **default**:  
                    cout << "Enter correct value\n";  
                    **break**;  
            }  
            **break**;  
        }  
    }  
}  
  
**void** **userChoiceValidation**() {  
    cout << "Enter your choice: ";  
    cin >> userChoice;  
}  
  
**void** **outputData**() {  
    revenueCalc();  
  
    sort(allTicketsDataSeason, allTicketsDataSeason + SEASON\_ARR\_LENGTH, [](**const** ticketsDataSeason& a, **const** ticketsDataSeason& b) {  
        **return** a.ticketsRevenue > b.ticketsRevenue;  
    });  
  
    cout << "Season  Price  Sold Tickets  Revenue\n";  
    **for** (**int** i = 0; i < SEASON\_ARR\_LENGTH; i++) {  
        cout<<allTicketsDataSeason[i].yearSeason;  
        printf("%12d", allTicketsDataSeason[i].price );  
        printf("%12d",allTicketsDataSeason[i].soldTicketsAmount);  
        printf("%12d\n",allTicketsDataSeason[i].ticketsRevenue);  
    }  
}

Результат работы программы

