Курсовая работа

по дисциплине: Системное программное обеспечение

на тему: «Использование библиотек в различных системах программирования»

I. Библиотеки подпрограмм как составная часть систем программирования

В состав системы программирования может входить большое количество разнообразных библиотек. Среди них всегда можно выделить основную библиотеку, содержащую обязательные функции входного языка программирования. Остальные библиотеки необязательны и подключаются к результирующей программе только по прямому указанию разработчика.

Новые возможности предоставили современные ОС, которые позволили подключать к результирующим программам не статические, а динамические библиотеки.

Динамические библиотеки в отличие от статических библиотек подключаются к программе не в момент ее компоновки, а непосредственно в ходе выполнения, как только программа затребовала ту или иную функцию, находящуюся в библиотеке. Преимущества таких библиотек — они не требуют включать в программу объектный код часто используемых функций.

Статические библиотеки – могут быть в виде исходного текста, подключаемого программистом к своей программе на этапе написания, либо в виде объектных файлов, присоединяемых (линкуемых) к исполняемой программе на этапе компиляции. В результате программа включает в себя все необходимые функции, что делает её автономной, но увеличивает размер.

Исходная программа

+

Статическая библиотека

=

Результирующая программа

Динамические библиотеки – это отдельные файлы, предоставляющие прикладным программам набор наиболее часто используемых функций, и загружаемые на этапе выполнения при обращении программы к ОС с заявкой на выполнение функции из библиотеки. Если запрошенная библиотека уже загружена в основное запоминающее устройство, программа будет пользоваться загруженной копией. Такой подход позволяет экономить память, поскольку несколько программ используют одну копию библиотеки, загруженную в память.

II. Включение текстов из файлов

Перечень обозначений заголовочных файлов для работы с библиотеками компилятора утвержден стандартом языка:

- **assert.h** - Диагностика программ
- **ctype.h** - Преобразование и проверка символов
- **errno.h** - Проверка ошибок
- **float.h** - Работа с вещественными данными
- **limits.h** - Предельные значения целочисленных данных
- **locate.h** - Поддержка национальной среды
- **math.h** - Математические вычисления
- **setjump.h** - Возможности нелокальных переходов
- **signal.h** - Обработка исключительных ситуаций
- **stdarg.h** - Поддержка переменного числа параметров
- **stddef.h** - Дополнительные определения
- **stdio.h** - Средства ввода-вывода
- **stdlib.h** - Функции общего назначения (работа с памятью)
- **string.h** - Работа со строками символов

Стандартные заголовочные файлы могут быть нечаянно или нарочно включены в текст программы в любом порядке и по несколько раз без отрицательных побочных эффектов. Однако действие включаемого заголовочного файла распространяется на текст программы только в пределах одного модуля от места размещения директивы **#include** и до конца текстового файла (и всех включаемых в программу текстов).

Заголовочные нестандартные файлы оказываются весьма эффективным средством при модульной разработке крупных программ, когда связь между модулями, размещаемыми в разных файлах, реализуется не только с помощью параметров, но и через внешние объекты, глобальные для нескольких или всех модулей. Описания таких внешних объектов (переменных, массивов, структур и т.п.) и прототипы функций помещаются в одном файле, который с помощью директив **#include** включается во все модули, где необходимы внешние объекты.

В тот же файл можно включить и директиву подключения файла с описаниями библиотеки функций ввода-вывода. Заголовочный файл может быть, например, таким:

```
#include<stdio.h>
/* Включение средств обмена */
/* Целые внешние переменные */
extern int ii, jj, 11;
/* Вещественные внешние переменные */
extern float aa, bb;
```

III. Библиотеки объектных модулей

Библиотека объектных модулей – это файл содержащий несколько объектных файлов, которые будут использоваться вместе в стадии присоединения к программе.

4. Создание статической библиотеки

Для создания статических библиотек существует простая специальная программа называемая **ar** (сокращенно от **archiver** – архиватор). Она используется для создания, модификации и просмотра объектных файлов в статических библиотеках, которые в действительности представляют из себя простые архивы.

Ключи программы **ar** имеют следующий смысл:

-d -исключить указанные (с помощью параметра *имя...)* файлы из архивного файла;

-r -заменить указанные (параметром *имя...)* файлы в архивном файле.

-t - вывести в стандартный поток вывода оглавление архивного файла.

-p - вывести в стандартный поток вывода указанные (параметром *имя...)* файлы из архива;

-v - выдавать пояснительные сообщения;

-x - извлечь из архива указанные (параметром *имя...)* файлы.

-c - создать архивный файл. Обычно программа ar при необходимости создает архивный файл сама. Данный ключ подавляет информационное сообщение, выдаваемое при создании архивного файла.

V. Создание библиотеки.

Рассмотрим создание библиотеки на примере нахождения значения функции:

$$z_1 = \frac{\sin(\frac{\pi}{2} + 3\alpha)}{1 - \sin(3\alpha - \pi)}$$

$$z_2 = ctg(\frac{5}{4}\pi + \frac{3}{2}\alpha)$$

Листинг программы:

```
#include <C:\TC\bib.ml>
#include <stdio.h>
#include <conio.h>
double zl (double a);
double z2 (double a);
main(){
double a;
clrscr();
printf("Vvedite znachiniye parametra a:");
scanf("%lf',&a);
printf("Znacheniye funckcii zl ravno: %lf\n", zl(a) );
printf("Znacheniye funckcii z2 ravno: %lf\n", z2(a) );
 printf("Programma zavershaet svoyu rabotu.\nDlya
    zakritiya najmite lubuyu klavishu.");
getch(); return 0;}
double zl (double a){
return (   ( sin( PI/2+3*a ) )/( l-sin( 3*a-PI ) )   );}
double z2 (double a){
return (   ctan( 5/4*PI+3/2*a )   );}
```

Создание библиотеки:

Библиотека создаётся в интегрированной среде программирования Borland Turbo C (tc.exe). Библиотека включает в себя функции, необходимые для вычисления значения z1 и z2 из задания. Также в библиотеку включена вещественная константа, значение которой приближается к значению π. Текст библиотеки приведён ниже.

Листинг библиотеки (bib.ml):

```
/*
This is a training library which include following function: sin(x) the x is double -> function return a double type cos(x) the x is double -> function return a double type ctan(x) the x is double -> function return a double type factorial(x) the x is double -> function return a double type pow(x,i) the x is double, i is int -> function return a double type and constants: PI-3,14159265358979
Pi=3,14159265358979323846264338327795 but it is too long :)
This library created by me: MC_CCCP for my laboratory works on SPE Data: 13042008
*/
```

```
/*Declaration*/
double sin (double x);
double cos (double x);
double ctan (double x);
 double factorial (double n );
double pow (double x,int i);
const double PI=3.14159265358979;
 /*Definition*/
double sin (double x){
double y=0;
int p=l,i;
for(i=l;i<50;i+=2){
y += p*pow(x,i)/factorial(i);
p*=-l;}
 return y;}
```

```
double cos (double x){
double y=0;
intp=l,i;
for(i=0;i<50;i+=2){
y += p*pow(x,i)/factorial(i);
p*=-l;}
return y;}
double ctan (double x){
return ( cos(x)/sin(x) ) ;}
double factorial (double n){
double f=l;
for (n;n>0;n--)
f*=n;
return f;}
double pow (double x, int i){
double y=l;
int j;
for (j=0; j<i; j++)
y*=x;
return y;}
```

5. Динамическая библиотека

Динамические библиотеки немного лучше статических, но их использование более сложное.

Объектный файл статистических библиотек вовсе не подходит для динамических библиотек. Связано это с тем, что все объектные файлы статистических библиотек не имеют представления о том, в какие адреса памяти будет загружена использующая их программа. Несколько различных программ могут использовать одну библиотеку, и каждая из них располагается в различном адресном пространстве. Поэтому требуется, чтобы переходы в функциях библиотеки (операции goto на ассемблере) использовали не абсолютную адресацию, а относительную. То есть генерируемый компилятором код должен быть независимым от адресов, такая технология получила название **PIC - Position Independent Code**. В компиляторе данная возможность включается ключом **-fPIC**.

6. Пример использования некоторых библиотек

Задача:

Написать программу, которая задумывает число в диапазоне от 1 до 10 и предлагает пользователю угадать число за 3 попытки.

```
#include <conio.h>
#include <stdlib.h>     // для доступа к srand
#include <time.h>
void main()
{
int comp;       // задуманное число
int igrok;      // вариант, игрока
int n;          //количество попыток
```

- time_t t; // текущее время - для инициализации генератора случайных чисел

```
srand((unsigned)   time(&t));
comp=rand()%10+1;
clrscr();
cprintf("\n\rKomputer\"zdumal\"chislo ot 1 do\10\n\r");
cprintf("Vi dolzni ego ugadat za 3 popitki");
```

```
n=0;
do {
cprintf("\n\r->");
cscanf("%i",&igrok);
n++;
}    while  ((igrok  != comp)&&(n < 3));
if  (igrok == comp)
{
textcolor (RED+BLINK);
cprintf("\n\r VI VIIGRALI!");
}
else
{
textcolor(GREEN);
cprintf("\n\r Vi proigrali");
cprintf ("Komputer zadumal chislo %d",comp);
}
textcolor(LIGHTGRAY);
cprintf("\n\r Dla zavershenia nazmite lubuu klavishu...");
getch();
getch();
}
```

В данной задаче использованы библиотеки conio.h, stdlib., time.h, их предназначение заключается в следующем:

Библиотека conio.h предназначена для работы с терминалом в текстовом режиме, в данной задаче используется функция textcolor(), которая устанавливает цвет символов по значению параметра, находящегося внутри круглых скобок (textcolor(GREEN)).

Библиотека stdlib.h предназначена для доступа к функции srand – функция инициализации генератора случайных чисел.

Библиотека time.h предназначена для определения дат и времени, в данной задаче предназначена для определения текущего времени для инициализации генератора случайных чисел.

VII. Язык программирования Python. Стандартная библиотека

Рассмотрим подробнее стандартную библиотеку Питона. Библиотека состоит из нескольких разделов:

1. Модули, дающие доступ к внутренностям интерпретатора и особенностям языка и реализации.
2. Модули для манипуляции со строками, в том числе и с юникодовыми строками.
3. Модули, дающие доступ к системной библиотеке, в первую очередь математические функции.
4. Модули для написания тестов в стиле Extreme Programming.
5. Модули для (относительно) переносимого способа доступа к

Так как модулей в стандартно библиотеке очень много, рассмотрим некоторые из них:

1. Первый важный модуль - **sys**, модуль доступа к Системе. Не операционной системе, а Системе. То есть к интерпретатору.

2. Модули **dumbdbm, dbm, gdbm, bsddb** дают доступ к соответствующим встраиваемым базам данных

3. **String -** модуль для манипуляции со строками без регулярных выражений. Здесь собраны функции для поиска подстроки в строке, замены и тому подобное и т. д.

IX. Библиотека TURBO VISION для TURBO PASCAL

Библиотека TURBO VISION предназначена для создания интерактивных программ, работающих в текстовом режиме, в соответствии со стандартом SAA/CUA .

Большинство программистов при написании каждой новой программы тратит много времени на разработку административной части.

TURBO VISION содержит элементы административной системы, которые наращиваются с помощью объектно-ориентированного программирования.

Пример программы на Turbo Pascal с использованием библиотеки TurboVision

Простейшая программа, написанная с использованием TURBO VISION, имеет вид:

```
Program Simplest;
uses App;
var MyApp : TApplication;
begin
 MyApp.Init;
 MyApp.Run;
 MyApp.Done;
end.
```

Завершается работа программы нажатием клавиш Alt+X или нажатием клавиши мыши, подведенной к строке состояния.

Эта программа, запущенная на выполнение, выводит на экран дисплея три подэлемента - пустую строку меню, рабочую область и строку состояния, которая содержит текст Alt+X Exit. Язык FORTRAN

Рассмотрим основные характеристики языка.

В заключении хочется сказать, что библиотеки подпрограмм входили в состав средств разработки, начиная с самых ранних этапов их развития. Даже когда компиляторы еще представляли собой отдельные программные модули, они уж были связаны соответствующими библиотеками, поскольку компиляция, так или иначе, предусматривает связь программ со стандартными функциями исходного языка. Эти функции обязательно должны входить в состав