ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

КАФЕДРА КИБЕРНЕТИКИ

ОДОБРЕНО  
  
протокол № 18 / 03   
  
от « 31 » мая 2018 г.

**РАБОЧАЯ ПРОГРАММА УЧЕБНОЙ ДИСЦИПЛИНЫ**

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

|  |  |
| --- | --- |
| Направление подготовки (специальность) | 09.04.04 Программная инженерия |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Семестр** | **Интерактив** | **Трудоемкость, кред.** | **Общий объем курса, час.** | **Лекции, час.** | **Практич. занятия, час.** | **Лаборат. работы, час.** | **СРС, час.** | **КСР, час.** | **Форма(ы) контроля, экз./зач./КР/КП** |
| 2 |  | 4 | 144 | 0 | 15 | 15 | 78 | 0 | Э |
| ИТОГО | 0 | 4 | 144 | 0 | 15 | 15 | 78 | 0 |  |

Группа: М20-504

АННОТАЦИЯ

В рамках дисциплины изучаются принципы объектно-ориентированного подхода в разработке программ.

1. ЦЕЛИ ОСВОЕНИЯ УЧЕБНОЙ ДИСЦИПЛИНЫ

Целями освоения учебной дисциплины «Объектно-ориентрованное программирование» являются изучение синтаксиса языка С++, понятий объектного и объектно-ориентированного подхода в разработке программ. Использованы алгоритмы курса численных методов.

Учебная задача: Получить опыт разработки, отладки конкретных алгоритмов численных методов в их реализации на языке С++ с использованием методологии объектно-ориентированного программирования.

2. МЕСТО УЧЕБНОЙ ДИСЦИПЛИНЫ В СТРУКТУРЕ ООП ВО

Дисциплина «Объектно-ориентрованное программирование» является обязательной дисциплиной инженерной подготовки студента. Дисциплина не требует специальной начальной подготовки, выходящей за рамки курса математики и информатики программы среднего образования.

В свою очередь, дисциплина является необходимым дополнением следующих курсов, демонстрируя методы и средства выполнения инженерных расчетов, связанных с общими методами математических вычислений, методами оптимизации, методами моделирования и проектирования сложных систем.

• дискретная математика

• логическое и функциональное программирование

• практикум на ЭВМ

• Основы автоматизированных информационных технологий

3. КОМПЕТЕНЦИИ СТУДЕНТА, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ОСВОЕНИЯ УЧЕБНОЙ ДИСЦИПЛИНЫ/ОЖИДАЕМЫЕ РЕЗУЛЬТАТЫ ОБРАЗОВАНИЯ И КОМПЕТЕНЦИИ СТУДЕНТА ПО ЗАВЕРШЕНИИ ОСВОЕНИЯ ПРОГРАММЫ УЧЕБНОЙ ДИСЦИПЛИНЫ

ОПК-2 – Способен разрабатывать оригинальные алгоритмы и программные средства, в том числе с использованием современных интеллектуальных технологий, для решения профессиональных задач

ОПК-5 – Способен разрабатывать и модернизировать программное и аппаратное обеспечение информационных и автоматизированных систем

ОПК-7 – Способен применять при решении профессиональных задач методы и средства получения, хранения, переработки и трансляции информации посредством современных компьютерных технологий, в том числе, в глобальных компьютерных сетях

ПК-12 – Способен проектировать вспомогательные и специализированные языки программирования и языки представления данных

ПК-19 – Владеет навыками создания систем обработки текстов

4. СТРУКТУРА И СОДЕРЖАНИЕ УЧЕБНОЙ ДИСЦИПЛИНЫ

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№ п.п** | **Наименование раздела учебной дисциплины** | **Недели** | **Лекции, час.** | **Практ. занятия / семинары, час.** | **Лабораторные работы, час.** | **Обязат. текущий контроль (форма\*, неделя)** | **Аттестация раздела (форма\*, неделя)** | **Максимальный балл за раздел\*\*** |
|  | *2 Семестр* |  |  |  |  |  |  |  |
| 1 | Элементы объектно-ориентированного программирования в современных языках программирования | 1-6 |  | 6 | 6 | к.р-6 | КИ-6 | 20 |
| 2 | Формализация ООП: Нетипизированные исчисления объектов | 7-8 |  | 2 | 2 | к.р-8 | КИ-8 | 30 |
| 3 | Формализация ООП: Типизированные исчисления объектов | 9-15 |  | 7 | 7 | к.р-15 | КИ-15 | 20 |
|  | *Итого за 2 Семестр* |  | 0 | 15 | 15 |  |  | 70 |
|  | **Контрольные мероприятия за 2 Семестр** |  |  |  |  |  | Э | 30 |

\* – сокращенное наименование формы контроля

\*\* – сумма максимальных баллов должна быть равна 100 за семестр, включая зачет и (или) экзамен

Сокращение наименований форм текущего контроля и аттестации разделов:

|  |  |
| --- | --- |
| **Обозначение** | **Полное наименование** |
| КИ | Контроль по итогам |
| к.р | Контрольная работа |
| Э | Экзамен |

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Недели** | **Темы занятий / Содержание** | **Лек., час.** | **Пр./сем., час.** | **Лаб., час.** |
|  | *2 Семестр* | 0 | 15 | 15 |
| **1-6** | **Элементы объектно-ориентированного программирования в современных языках программирования** |  | 6 | 6 |
| 1 - 2 | **Основные понятия и механизмы программирования и языков С. С++.** Специфика иерархического, объектного и объектно-ориентированного подходов в разработке программ. Схема по-строения приложений в объектно-ориентированной методологии проектирования. | Всего аудиторных часов | Всего аудиторных часов | Всего аудиторных часов |
|  | 2 | 2 |
| Онлайн | Онлайн | Онлайн |
|  |  |  |
| 3 - 4 | **Синтаксиса языка С++. Переход синтаксиса от С к С++.** Синтаксис С++. Механизм защиты приложений. Конструирование объектов. Описание функций. Наследование и виртуальные определения. | Всего аудиторных часов | Всего аудиторных часов | Всего аудиторных часов |
|  | 2 | 2 |
| Онлайн | Онлайн | Онлайн |
|  |  |  |
| 5 - 6 | **Элементы объектно-ориентированного программирования.** Понятие и определение класса и его методов. Конструктор класса. Наследование. Перегрузка операций. Полиморфизм описаний. | Всего аудиторных часов | Всего аудиторных часов | Всего аудиторных часов |
|  | 2 | 2 |
| Онлайн | Онлайн | Онлайн |
|  |  |  |
| **7-8** | **Формализация ООП: Нетипизированные исчисления объектов** |  | 2 | 2 |
| 7 - 8 | **Использование шаблонов при описании объектов.** Понятие «шаблона» и его использование при программировании. Абстракции данных и процедур. | Всего аудиторных часов | Всего аудиторных часов | Всего аудиторных часов |
|  | 2 | 2 |
| Онлайн | Онлайн | Онлайн |
|  |  |  |
| **9-15** | **Формализация ООП: Типизированные исчисления объектов** |  | 7 | 7 |
| 9 - 11 | **Обработка исключений.** Синтаксис обработки ситуаций обработки исключений. Примеры описания и использования. | Всего аудиторных часов | Всего аудиторных часов | Всего аудиторных часов |
|  | 3 | 3 |
| Онлайн | Онлайн | Онлайн |
|  |  |  |
| 12 - 15 | **Разбор конкретных примеров использования методологии объектно-ориентированного программирования** Разбор конкретных примеров использования методологии объектно-ориентированного программирования | Всего аудиторных часов | Всего аудиторных часов | Всего аудиторных часов |
|  | 4 | 4 |
| Онлайн | Онлайн | Онлайн |
|  |  |  |

Сокращенные наименования онлайн опций:

|  |  |
| --- | --- |
| **Обозначение** | **Полное наименование** |
| ЭК | Электронный курс |
| ПМ | Полнотекстовый материал |
| ПЛ | Полнотекстовые лекции |
| ВМ | Видео-материалы |
| АМ | Аудио-материалы |
| Прз | Презентации |
| Т | Тесты |
| ЭСМ | Электронные справочные материалы |
| ИС | Интерактивный сайт |

ТЕМЫ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

|  |  |
| --- | --- |
| **Недели** | **Темы занятий / Содержание** |
|  | *2 Семестр* |
| 1 - 6 | **Элементы объектно-ориентированного программирования** Элементы объектно-ориентированного программирования |
| 7 - 8 | **Использование шаблонов при описании объектов** Использование шаблонов при описании объектов |
| 9 - 17 | **Разбор конкретных примеров использования методологии объектно-ориентированного программирования** Разбор конкретных примеров использования методологии объектно-ориентированного программирования |

5. ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ

Практические занятия проводятся по традиционной схеме, на отдельных практических занятиях студенты выступают с докладами и проводят их обсуждение. На практических занятиях регулярно проводится контрольный опрос для проверки освоения студентами теоретического материала, предусмотрена контрольная работа в компьютерном классе.

6. ТРЕБОВАНИЯ К ФОНДУ ОЦЕНОЧНЫХ СРЕДСТВ В РАМКАХ РЕАЛИЗУЕМОЙ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ

В качестве оценочного средства используется 100 бальная семестровая система, учитывающая выполнение тематических домашних заданий по каждому разделу и контрольной работа по каждому разделу. Каждый раздел считается «зачтенным» на указываемую бальную оценку, если полностью выполнено домашнее задание и данная тема «защищена» на соответствующей контрольной работе.

ТДЗ\_1 – выполнения тематического ДЗ\_1

Выполнено +10 баллов

Не выполнено 0 баллов по всей теме независимо от ТКР

КТР\_1 - контрольная работа (продолжительность – 1 а/час (проводится в аудитории)

Выполнено не менее 90% Зачтены баллы ТДЗ (полностью)

Выполнено от 90-до 50% 0.5 от баллов ТДЗ

Менее 50% - общий незачет темы (ТДЗ и ТКЗ) - 0 баллов

ТДЗ\_2 – выполнения тематического ДЗ\_2

Выполнено +10 баллов

Не выполнено 0 баллов по всей теме независимо от ТКР

КТР\_2 - контрольная работа (продолжительность – 1 а/час (проводится в аудитории)

Выполнено не менее 90% Зачтены баллы ТДЗ (полностью)

Выполнено от 90-до 50% 0.5 от баллов ТДЗ

Менее 50% - общий незачет темы (ТДЗ и ТКЗ) - 0 баллов

ТДЗ\_3 – выполнения тематического ДЗ\_3

Выполнено полностью +30 баллов

Выполнена проектная часть. Отлажена. Отсутствует тестирование.

+20 баллов

Выполнена проектная часть. Не отлажена

+10 баллов

Не выполнено 0 баллов

КИ – аттестация раздела (контроль по итогам раздела):

Раздел аттестуется, если выполнено ТДЗ и соответствующая КТР. При невыполнении какой-либо компоненты (ТДЗ или ТКР) раздел не зачитывается полностью.

Зачет предполагает обязательное предоставление материалов разработанных по всем темам семестра.

«Зачет» по курсу выставляется при сумме полученных баллов за семестр – не менее 50.

Собеседование на зачете может дополнительно 0-50 баллов.

ТДЗ принимаются в течение всего семестра.

КТЗ пересдаются на зачетной неделе. Организуется не менее 3 пересдач.

Зачет организуется как собеседование по итогам проделанной работы за семестр. Рассматриваемые материалы – все выполненные ТДЗ и КТР.

Основные тематические примеры и фрагменты раздаточного материала.

Комментарии

В Си++ можно использовать два вида комментариев:

обычные, оформленные по правилам Си,

и однострочные, начинающиеся с символов // и продолжающиеся до конца строки.

//Это однострочный комментарий

/\*Это комментарий,

продолжающийся на другой строке\*/

/\*Коммен-/\*Такие вложенные комментарии обычно недопустимы\*/- тарий\*/

/\*Коммен-//А вот такое вложение возможно! -тарий\*/

//Коммен-/\*И так тоже можно!\*/тарий

В редких случаях при комментировании бывают недоразумения, например:

х=у//\*это деление\*/z; Достаточно добавить пробел, чтобы избежать двусмысленности:

х=у/ /\*это деление\*/z;

Листинг 1. Локальные переменные внутри блока.

//расширение 'срр' принято для файлов с текстами программ на Си++

#include <stdio.h>

int main()

{

//В Си++ "модно" таким образом описывать и присваивать начальное

//значение счетчику цикла for

for (int counterl=0; counterl<3; counterl++)

// Переменная counteri1 "видна", начиная с этой строки и до конца main,

// (а не только внутри блока for!)

// Ей присваивается значение 0 перед входом в цикл.

(

//Автоматической переменной i ПРИСВАИВАЕТСЯ значение 0 при

//КАЖДОМ проходе тела цикла,

int i=0;

//а статическая переменная j ИНИЦИАЛИЗИРУЕТСЯ нулем!

static int j=0;

for (int counter2=0; counter2<10; counter2++)

printf("i=%d j=%d\n",i++, j++);

}

//counter2 "существует" вот до этой фигурной скобки.

char quit\_message[]="Вуе!";

printf("%s", quit\_message);

return 0;

}

Строка из программы, приведенной ранее:

int i=0;

полностью эквивалентна сочетанию описания и оператора присваивания:

int i ;

i = 0;

Она описывает локальную в охватывающем блоке автоматическую переменную

и осуществляет присваивание этой переменной значения 0 при каждом проходе

тела цикла.

Выражение в правой части присваивания, разумеется, может и не быть константным.

В то же время запись

static int j=0; не эквивалентна

static int j; j=0;

Такая запись описывает локальную в охватывающем блоке статическую переменную и инициализирует ее нулем — иными словами, j будет иметь значение 0 при первом входе в охватывающий блок.

В этом случае инициализирующее выражение обязательно должно быть константным. Однако возможны неожиданности: попробуйте, например, в программе из листинга 1 заменить

static int j=0; на

static int j=i + i; //не константное выражение в

//правой части!

Результат зависит от используемого компилятора.

Borland C++ 3.0, не выдавая никаких предупреждений, инициализирует j нулем,

а присваивания при каждом проходе цикла делать не будет.

Так что будьте внимательны.

А вот пример фрагмента программы с синтаксическими ошибками:

int k;

k=int i=0; //Ошибка!

printf (" %d" , int j = 1; ) ; //Ошибка!

for (int count=0; count<max\_count; count++)

{

//операторы...

}

//ОШИБКА! Повторное описание count

for (int count=0; count<max\_count; count++)

{

//операторы...

}

Листинг 2. Использование прототипов функций.

//header.h (иногда заголовочным файлам для

//программ на Си++ дают расширение 'hpp')

void func (int, double\*);

//void означает, что функция не возвращает никакого значения;

//при желании можно написать и так: void func1(int j, double\* x\_ptr);

//имена аргументов в прототипах необязательны, но их наличие

//облегчает чтение листингов.

double func2();

//Внимание! В отличие от Си, где пустые скобки в описании функции

//означают произвольное количество аргументов любого типа, в Си++

//они означают отсутствие передаваемых аргументов (void).

void func3(int, ...);

//Первый аргумент - типа int, тип и количество остальных могут

//варьироваться.

//mod1.cpp

#include "header.h"

...

int main()

{

int i ;

double x[20], y;

...

funcl(i, x);

y=func2();

func3(i, x, func2);

...

//mod2.cpp

//Определения функций, вызываемых в modi.cpp.

//Обратите внимание, что заголовки функций в определениях также

//оформляются "по-новому".

//ПРИМЕЧАНИЕ: некоторые компиляторы Си++ допускают заголовки

//типа int f(i, j) int i,j; {тело функции}, но большинство - нет.

...

void func1(int i, double \*x)

{

//тело функции...

}

double func2()

{

...

}

void func3(int i, ...)

{

...

}

В прототипах функций можно также задавать значения аргументов,

передаваемые по умолчанию. Предположим, вы написали функцию Draw/Circle, которая рисует на экране окружность данного радиуса с центром в данной точке, и задали ее прототип:

void DrawCircle(int x=100, int y=100, int. radius=100); или просто:

void DrawCircle(int=100, int=100, int=100);

хотя так, по-моему, труднее читается.

Тогда вызовы этой функции будут проинтерпретированы,

в зависимости от количества передаваемых аргументов, следующим образом:

DrawCircle (); //рисуется окружность с центром

//в точке (100, 100) и с радиусом 100

DrawCircle (200) ; //центр в точке (200, 100), радиус 100

DrawCircle (200,300);//центр в точке (200, 300),

//радиус 100

DrawCircle (200,300,400); //центр в точке (200, 300),

//радиус 400

DrawCircle (, ,400); //Ошибка! Разрешается опускать

//аргументы только справа!

void DrawCircle(int x, int y=200, int radius); // Ошибка!

и еще

void func(double\*=0); // Ошибка! \*= является присваиванием

void func(double\* =0); // так правильно

Доступ к глобальным переменным, скрытым локальными переменными с тем же именем.

Оператор разрешения области видимости ::

(scope resolution operator — двойное двоеточие)

позволяет воспользоваться глобальной переменной в случае,

если она скрыта локальной переменной с тем же именем:

int i=0; //глобальная переменная i

...

int f()

{

...

int i=0; //локальная переменная i внутри функции f

i++; //увеличивается локальная i

::i++; //увеличивается глобальная i

...

}

Листинг 3. Функциональная запись преобразования типа.

int a, b;

typedef char\* PChar;

typedef void\* PVoid;

typedef int\* Pint;

PChar c\_ptr, d\_ptr;

PVoid v\_ptr;

//...

a= (int \*) v\_ptr; //операторная запись

b=PInt (v\_ptr) ; //функциональная запись

c\_ptr= (PChar) v\_ptr;//операторная

d\_ptr=PChar (v\_ptr) ;//функциональная

v\_ptr=PVoid (PChar (v\_ptr)+1) ; //выглядит явно лучше,

//чем следующая строка!

v\_ptr=(void\*)((char\*)v\_ptr+l);Листинг 4. Модификатор const делает переменную недоступной в других модулях.

//Первый модуль.

//modi.cpp

const float pi=3.14159;

//Второй модуль.

//mod2.cpp

#include <stdio.h>

extern float pi;

int main()

{

printf("%f", pi);

return 0;

}

Модификатор volatile, напротив, сообщает компилятору,

что значение данной переменной может быть изменено

каким-либо фоновым процессом — например, при обработке прерывания.

С точки зрения компилятора это означает, что вычисляя значение выражения,

в которое входит такая переменная, он должен брать ее значение только из памяти

(а не использовать копию, находящуюся в регистре, что допустимо в других случаях).

Применение const и volatile распространяется в Си++ также на классы и функции-члены.

Листинг 5. Передача параметров по значению и по ссылке.

#include <stdio.h>

void swapl(int x, int y)

{

printf("внутри swap1 перед обменом x=%d, y=%d\n", x y);

int z=y;

У=х;

x=Z ;

printf("внутри swap1 после обмена x=%d, y=%d\n", x, y);

}

void swap2(int \*x, int \*y)

{

printf ("внутри swap2 перед обменом \*x=%d, \*y=%d\n", \*x, \*y);

int z=\*y;

\*y=\*x;

\*x=z ;

printf("BHyTpn swap2 после обмена \*x=%d, \*y=%d\n", \*x, \*y);

}

void swap3(int& x, int& y)

//& x означает, что х будет передаваться по ссылке, то есть

//воздействие на формальные параметры внутри swap3 приведет к

//изменению значений аргументов, с которыми swap3 вызвана ,а в

//остальном swap3 ничем не отличается от swap1!

//ПРИМЕЧАНИЕ: type & читается как "ссылка на тип type".

{

printf ("внутри swap3 перед обменом x=%d, y=%d\n", x , у);

int z=y;

У=х;

X=Z ;

printf ("внутри swap3 после обмена x=%d, y=%d\n", x, у);

}

void swap4(double &x, double &y)

{

printf ("внутри swap4 перед обменом x=%f, y=%f\n", x, у);

int z=y;

У=х;

x = z;

printf ("внутри swap4 после обмена x=%f, y=%f\n", x, y);

}

int main()

{

int a=0, b=l;

printf ("первоначально a=%d, b=%d\n", a, b);

swapl(a, b);

printf ("после swapl a=%d, b=%d\n", a, b);

swap2(&a, &b);

printf(¦'пocлe swap2 a=%d, b=%d\n", a, b);

swap3(a, b);

printf("пocлe swap3 a=%d, b=%d\n", a, b);

swap4(a, b);

printf ("после swap4 a=%d, b=%d\n", a, b);

return 0;

}

#include <stdio.h>

#include <conio.h>

int f(int i){return ++i;}

int g(int &i){return ++i;}

int h(char &i){return ++i;}

main()

{

int a=0,b=0,c=0;

a += f(g(a)); // a+ = f(g(a)); error!

b += g(f(b));

c += f(h(c));

printf("\n a=%d, b=%d, c=%d",a,b,c);

getch();

}

// a=3, b=2, c=2

Листинг 6. Передача структур по ссылке.

#include <stdio.h>

#include <string.h>

struct Student

{

char FirstName[20];

char SecondName[20];

char Surname[20];

int Age;

char Dept[20] ;

int Year;

//имя

//в данном случае отчество

//фамилия

//возраст

//факультет

//курс

}

//Распечатать информацию о студенте.

//Обратите внимание, что функция возвращает ссылку на Student, что

//позволит в дальнейшем использовать ее в левой (!) части оператор\*

//присваивания.

Student& PrintStudentlnfo(Student& st)

{

printf("Firstname %s\n"

"Secondname %s\n"

"Surname %s\n"

"Age %d\n"

"Department %s\n"

"Year %d\n"

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

st.FirstName,

st. SecondName,

st.Surname,

st.Age,

st.Dept,

st.Year); return st;

}

//сократить имя и отчество до инициалов

Student& ShortenNames(Student& st)

{

st.SecondName[1]=st.FirstName[1] ='.';

st.SecondName[2]=st.FirstName[2]='\x0';

return st;

}

//записать фамилию заглавными буквами

Student& Capitalize(Student& st)

{

strupr(st.Surname); //прототип в string.h

return st;

}

int main()

//B Си++ не обязательно использовать ключевое слово struct при

//описании переменных структурного типа.

Student stl ={

"Ivan",

"Ivanovitch",

"Sidorov",

24,

"Chemistry",

3

};

Student st2={

"Sidor",

"Sidorovitch",

"Ivanov",

26,

"Math",

5

};

PrintStudentlnfo(stl);

Student stl\_mod=Capitalize(ShortenNames(stl));

//можно было и наоборот:

//ShortenNames(Capitalize(stl));

PrintStudentlnfo(stl\_mod) = st2 ; //С точки зрения

//"обычного" Си это выражение - ошибка,а для Си++

//ничего необычного: PrintStudentlnfo возвращает ссылку на st1\_mod,

//а присвоение значения ссылке эквивалентно присвоению значения

//тому объекту, на который она ссылается (в данном случае st1\_mod).

PrintStudentlnfo(stl\_mod) ; //st1\_mod теперь имеет то

//же значение, что и st2

return 0;

}

Использование псевдонимов (aliases)

int x=l;

int &xr=x; //ссылка хr становится псевдонимом x

xr=2; //все равно, что х=2,

//так как хг полностью эквивалентна х

хг++; //то же, что и х++

Однако,

int х=1;

char &xr=x; //Дак как типы х и r не совпадают,

//компилятор создает временную переменную типа char, для

//которой хr будет псевдонимом, и присваивает ей значение

//(char)x, т. е. '\х1'.

хr=2; //значение х не изменяется!

Листинг 7. Замена функции макрокомандой подчас может

обернуться трудноуловимой ошибкой.

#include <stdio.h>

int FunctionCube(int x)

{ return x\*x\*x; }

#define MacrosCube(x) ((x)\*(x)\*(x))

int main()

{

int a = 2;

printf("FunctionCube(a++)= %d, ",

FunctionCube(a++));

printf("a=%d\n", a) ;

printf("MacrosCube(а++)=%d, ", MacrosCube(a++));

printf("a=%d\n", a);

return 0;

}

В Си++ достаточно описать функцию как inline — и компилятор,

если это возможно (сравните с описанием переменной как register!),

будет подставлять в соответствующих местах тело функции, вместо того,

чтобы осуществлять ее вызов.

Конечно, определение подставляемой функции должно находиться перед

ее первым вызовом:

inline int InlineFunctionCube(int x)

{

return x\*x\*x;

b=InlineFunctionCube(a);

c=InlineFunctionCube(a++);

... Листинг 8. Использование операторов new и delete.

#include <stdio.h>

int main()

{

int \*i\_ptr;

double \*d\_ptr;

char \*string;

int str\_len=80;

i\_ptr=new int; //Зарезервировать место под переменную

//типа int и присвоить i\_ptr ее адрес. Блок памяти будет занят

//до соответствующего оператора delete или, при его

//отсутствии, до конца программы.

//Значение \*i\_ptr не определено!

d\_ptr=new double (3.1415); //Аналогично предыдущему,

//только \*d\_ptr инициализируется значением 3.1415

string-new char [str\_len] ;//Отвести место в свободной

//памяти под массив из str\_len элементов типа char,string

//теперь содержит адрес нулевого элемента массива. Если

//выделить память невозможно, то оператор new возвращает

//значение NULL, т.е. (void\*)0

if (!(i\_ptr && d\_ptr && string))

printf("He хватает памяти для всех динамически "

"размещаемых переменных!");

return 1;

}

string[0]='Н'; string[1]='i'; string[2]='i'; string[3]='\x0';

printf("i\_ptr=%p случайное значение \*i\_ptr=%d\n",i\_ptr,\*i\_ptr);

delete i\_ptr;//освободить блок памяти, на который

//указывает i\_ptr. Примечание: если указатель, на который

//действует оператор delete, не содержит адрес блока,

//зарезервированного ранее оператором new, или же не равен

//NULL, то последствия будут непредсказуемыми!!!

printf("d\_ptr=%p \*d\_ptr=%f\n", d\_Ptr,\*d\_ptr);

delete d\_ptr;

printf ("string=%p string contents=%s\n",string,string)

delete[str\_len] string;

//можно и так: delete string; компилятор поймет...

return 0;

}

В Си++ можно написать свою функцию "поиска" свободной памяти.

Соответственно и ее управление (чтобы не циклилась).

Т.е. функция пытается зарезервировать требуемую память, и,

если ее по-прежнему не хватает, программа зациклится.

Чтобы установить определенную пользователем функцию работки, необходимо присвоить

ее адрес указателю \_new\_handler, объявленному в стандартном заголовочном файле

new.h как

typedef void (\*pvf) () ; //указатель на функцию без аргументов

//не возвращающую значения

extern pvf \_new\_handler;

Другой способ состоит в вызове библиотечной функции set\_new\_handler,

прототип которой также находится в new.h

pvf set\_new\_handler(pvf);

Аргумент функции set\_new\_handler — адрес устанавливаемой функции-обработчика,

возвращаемое значение — адрес старого обработчика.

Листинг 9. На Си каждой из функций для вывода переменных

разного типа придется дать особое имя.

void print\_int(int i)

{printf("%d", i);}

void print\_double(double x)

{printf ("%f\ x);}

void print\_string(char\* s)

{printf("%s", s);}

int j=5; print\_int(j ) ;

print\_double(3.141592);

print\_string("Hi, there!");

Листинг 10. Перегрузка функций в Си++.

#include <stdio.h>

void print(int i)

{printf("%d "„ i);}

void print(double x)

{printf("%f ", x);}

void print(char\* s)

{printf ("%S '., S) ;}

int main()

{

int j=5;

double e=2.7183;

float pi=3.1415926;

print (j ) ;

print(e) ;

print(pi)

print("Hi!";

return 0;

)

Перегруженные функции не могут различаться только по типу возвращаемого значения:

void f(int, int);

int f(int, int); //ОШИБКА!

Перегрузка функций не должна приводить к конфликту с аргументами, заданными по умолчанию:

...

void f(int = 0); //прототипы

void f();

f( ) ; //КАКУЮ ФУНКЦИЮ ВЫЗВАТЬ?

Поскольку компилятор Си++ позволяет давать различным функциям одинаковые имена, то, помещая имена функций в объектный файл — результат компиляции, — он должен их каким-либо образом модифицировать, чтобы сделать уникальными. Очевидно, модифицированные имена должны содержать информацию о количестве и типе аргументов, так как именно по этому признаку перегруженные функции различаются между собой. Такая модификация получила название «искажение имен» (name mangling). В некоторых ситуациях, например, при необходимости скомпоновать программу на Си++ с объектными файлами или библиотеками, созданными «обычным» Си-компилятором, искажение имен нежелательно. Чтобы сообщить компилятору Си++, что имена тех или иных функций не должны искажаться, их следует объявить как extern "С":

extern "С" int funcl(int); //отдельная функция extern "С"

//несколько функций

{

void func2(int);

int func3();

double func4(double);

};

Модификатор extern "С" можно использовать не только при описании, но и при определении функций:

extern "С" int my\_func(int i)

//my\_func можно вызвать в модуле, написанном на Си

{

//тело функции...

}

Функции, описанные как extern "С", не могут быть перегруженными.

Листинг 11. Шаблоны функций.

#include <string.h>

#inciude <stdio.h>

//Компилятор создаст подходящую функцию когда "узнаёт"

//какой тип аргументов Т подходит в конкретном случае.

//Новое ключевое слово "class":

//Т может быть и именем простого типа данных.

template <class T> void swap(T &a,T &b)

{

Т с;//создать переменную для временного хранения значени!

c=b;b=a;а=с; //обменять

};//здесь символ ";" не обязателен

int main()

(

int i=0, j=l;

double x=0.0, y=1.0;

char \*sl="Hi,I am the first string!",

\*s2="Hi,I am the second string!";

printf("Перед обменом:\n"

"i=%d j=%d\n"

"x=%f y=%f\n"

"sl=%s s2=%s\n",

i,j,x,y,s1,s2);

swap(i,j);

swap(x,у);

swap(si,s2);

printf("После обмена:\n"

"i=%d j=%d\n"

"x=%f y=%f\n"

"sl=%s s2=%s\n",

i,j,x,y,si,s2);

return 0;

}

Листинг 12. Перегрузка операторов.

#include <stdio.h>

#include <string.h>

const MAX\_STR\_LEN=80;

struct String //структурный тип Строка (с заглавной буквы)

{

char s[MAX\_STR\_LEN]; //массив символов -

//"содержимое" Строки

int str\_len; //текущая длина Строки

};

//переопределим ("перегрузим") оператор сложения

//для данных типа String

String operator+(String si, String s2)

{

String TmpStr; //для временного хранения...

//Длина Строки-результата сложения равна сумме длин

//складываемых Строк. Позаботимся также о том, чтобы не

//выйти за границу отведенного массива.

if ((TmpStr.str\_len=sl.str\_len+s2.str\_len)

>=MAX\_STR\_LEN)

{

TmpStr.s[0]='\x0';

TmpStr.str\_len=0;

return TmpStr;

}

//а теперь сольем Строки

strcpy(TmpStr.s, sl.s);

strcat(TmpStr.s, s2.s);

return TmpStr;//возвратим результат

int main()

String strl, str2, str3 ;//Еще раз заметим, что при объявлении

//переменных типа struct ключевое слово struct можно не

//использовать. "Начиним" strl и str2 содержимым.

strcpy(strl.s,"Перегрузка операторов - ");

strl.str\_len=strlen(strl.s) ;

strcpy(str2.s,"это прекрасно!"};

str2.str\_len=strlen(str2.s);

printf ("Первая Строка: длина=%d,содержимое=%з\п",

strl.str\_len,strl.s);

printf ("Вторая Строка: длина=%d,содержимое=%з\п",

str2.str\_len,str2.s);

str3=strl + str2 ;//Используем перегруженный оператор!

//Компилятор, ориентируясь на типы слагаемых, генерирует код,

//эквивалентный вызову str3=operator+(str1,str2);

printf("Третья Строка: длина=%d,содержимое=%з\п",

str3.str\_len,str3.s);

return 0;

}

// Простой пример обработки исключительных ситуаций

#include <iostream.h>

main()

{

cout << "начало\n";

try { // начало блока try

cout << "Внутри блока try\n";

throw 10; // генерация ошибки

cout << "Это выполнено не будет";

}

catch (int i) { // перехват ошибки

cout << "Перехвачена ошибка номер: ";

cout << i << "\n";

}

cout << "конец";

return 0;

}

// Этот пример работать не будет

#include <iostream.h>

main()

{

cout << "начало\n";

try { // начало блока try

cout << "Внутри блока try\n";

throw 10; // генерация ошибки

cout << "Это выполнено не будет";

}

catch (double i) { // не будет работать с исключительной ситуацией

// целого типа

cout << "Перехвачена ошибка номер: ";

cout << i << "\n";

}

cout << "конец";

return 0;

}

// Генерация исключительной ситуации из функции, находящейся вне блока try

#include <iostream.h>

void Xtest(int test)

{

cout << "Внутри Xtest, test равно: " << test << "\n";

if(test) throw test;

}

main()

{

cout << "начало\n";

try { // начало блока try

cout << "Внутри блока try\n";

Xtest(0);

Xtest(1);

Xtest(2);

}

catch (int i) { // перехват ошибки

cout << "Перехвачена ошибка номер: ";

cout << i << "\n";

}

cout << "конец";

return 0;

}

//начало

//Внутри блока try

//Внутри Xtest, test равно: 0

//Внутри Xtest, test равно: 1

//Перехвачена ошибка номер: 1

#include <iostream.h>

// Блоки try и catch могут находиться в функции, отличной от main()

void Xhandler(int test)

{

try {

if(test) throw test;

//When an exception occurs, the throw expression initializes a temporary object //of the type T

}

catch(int i) {

cout << "Перехвачена ошибка номер: " << i << '\n';

}

}

main()

{

cout << "начало\n";

Xhandler(1);

Xhandler(2);

Xhandler(0);

Xhandler(3);

cout << "конец";

return 0;

}

//начало

//Перехвачена ошибка номер: 1

//Перехвачена ошибка номер: 2

//Перехвачена ошибка номер: 3

//конец

#include <iostream.h>

// Можно перехватывать разные типы исключительных ситуаций

void Xhandler(int test)

{

try {

if(test) throw test;

else throw "Значение равно нулю";

}

catch(int i) {

cout << "Перехвачена ошибка номер: " << i << '\n';

}

catch(char \*str) {

cout << "Перехвачена стока: ";

cout << str << '\n';

}

}

main()

{

cout << "начало\n";

Xhandler(1);

Xhandler(2);

Xhandler(0);

Xhandler(3);

cout << "конец";

return 0;

}

//начало

//Перехвачена ошибка номер: 1

//Перехвачена ошибка номер: 2

//Перехвачена стока: Значение равно нулю

//Перехвачена ошибка номер: 3

//конец

7. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ УЧЕБНОЙ ДИСЦИПЛИНЫ

ОСНОВНАЯ ЛИТЕРАТУРА:

1. ЭИ Б 87 Вопросы и упражнения на понимание физики : , Москва: Лань, 2017

2. ЭИ П 30 Моделирование процессов и систем : учебное пособие, Санкт-Петербург: Лань, 2015

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА:

1. 004 С83 Программирование. Принципы и практика использования C++ : , Москва [и др.]: Вильямс, 2011

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ:

Специальное программное обеспечение не требуется

LMS И ИНТЕРНЕТ-РЕСУРСЫ:

-

8. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ УЧЕБНОЙ ДИСЦИПЛИНЫ

Специальное материально-техническое обеспечение не требуется

9. ##Definition not found: 'static\_section\_edu\_stud'##

Основное назначение данного раздела - выработка у студентов навыков правильного оформления документации по проекту, в данном случае, тем задачам, которые входят в объем задания практикума. Это стимулирует:

- самостоятельное изучение языков программирования,

- ясное представление и соответствующее оформление постановки решаемой задачи,

- четкий выбор и обоснование изобразительных средств используемого инструментального языка про-граммирования.

Каждому студенту предлагается набор из 3 - 4 задач общего задания, по которым он должен предста-вить помимо программной реализации на выбранных (или указанных преподавателем) языках (Паскаль, Модула-2, Си, С++) полную проектную документацию.

Первым этапом подготовки документации является разработка достаточно подробной спецификации для каждой выбранной задачи. Спецификация оформляется и согласуется с ведущим преподавателем до начала программирования (кодирования) и не должна зависеть от особенностей языка реализации. Допускается на данном этапе производить оформление основной логической схемы реализации в виде текста на псевдокоде.

Спецификация должна включать в себя требования к проверке правильности функционирования программной реализации. Требования должны покрывать все основные функции программной реализации (ввод, обработку ошибок, вычисление результата и т.п.).

Вторым этапом решения задачи является составление программы (программ) на языке (языках) реализации. При этом основное внимание должно быть уделено точности соблюдения требований спецификации и обоснованию используемых средств инструментального языка.

Если в процессе реализации произведены уточнение и(или) декомпозиция требований по проверке правильности функционирования программной реализации, то рекомендуется оформить это в виде дополни-тельных требований по проверке реализации.

Дальнейшим этапом является составление тест-плана - программы испытаний (проверки, тестирова-ния) программной реализации. Критерием качества тест-плана является покрытие (выполнение) всех тре-бований к проверке правильности функционирования программной реализации. Желательной характеристи-кой тест-плана является проверка исполнения всех веток схемы программной реализации. Тест-план должен быть оформлен и согласован с ведущим преподавателем в процессе исполнения программной реализации задачи.

Заключительным этапом работы является исполнение тест-плана, и предъявление результатов преподавателю. При приеме (защите результатов) домашнего задания преподаватель должен проверить степень обоснованности проектных решений, принятых при программной реализации задачи. С этой целью рекомендуется рассмотреть и попросить студента обосновать предпочтение выбора из нескольких вариантов представления данных, возможных альтернатив организации вычислительного процесса и конструкций языка реализации.

Далее приведена рекомендуемая структура оформления спецификации задачи.

Задание

1. Постановка задачи

Написать программу, выполняющую кодирование входной строки перестановкой символов по правилу 2-3-1.

Первый этап работы.

Необходимо представить поведение программы в целом. Необходимо ли получить программу кодирования одной единственной строки или требуется диалоговый режим управления кодированием группы строк, каждая из которых вводится после некоторого приглашающего сообщения?

Можно предположить, что из общих соображений простоты общения и проверки работоспособности второй вариант (циклическая программа) представляется предпочтительным. Соответственно, первый раздел требований (спецификаций) может быть сформулирован следующим образом:

1. Необходимо разработать программу, выполняющую следующие функции:

1.1. Ввод строки символов с клавиатуры.

1.2. Преобразование входной строки путем перестановки символов в порядке 2-3-1.

1.3. Вывод преобразованной строки на экран.

1.4. Программа должна обрабатывать строки последовательно, каждый раз предупреждая пользователя о необходимости очередного ввода. Обработка должна быть эакончена при вводе пользователем пустой строки.

Второй этап работы.

Необходимо принять решения по основным позициям внешнего проекта программы.

Что такое входная строка?

(Последовательность символов, ограниченная входным алфавитом.)

Существенно ли деление строки на слова?

(Пробел, как специальный символ алфавита разделитель/ограничитель слов.)

Может ли в строку (в слово) быть включен символ не имеющий адекватного представления на клавиатуре или символ-разделитель, как часть слова?

Существует ли ограничение на длину строки?

Как определить, что входная строка введена полностью?

(Ограничитель строки и признак конца ввода.)

Как выходная (преобразованная) строка сопоставляется с входной?

(Формат вывода, размещение строки на экране.)

Как производится преобразование при длине входной строки не кратной трем?

Каждый автор может принять собственное решение по данным пунктам. В любом случае свои решения в рамках домашнего практикума необходимо согласовать с преподавателем.

В рассматриваемом далее варианте внешнего проекта предлагается несколько упрощенный вариант требований, предполагающий разбивку строки на слова и выполнение заданных правил перестановки только в пределах отдельного слова. При этом не накладывается сложных ограничений на разделитель слов. Им может быть многоточие или произвольная последовательность точек, запятых и пробелов.

2. Внешний проект

2.1. Входные данные программы.

2.1.1. Входная строка должна вводиться с клавиатуры после вывода на экран приглашающего сообщения.

2.1.2. Входная строка представляет собой последовательность слов, разделенных (ограниченных) символами разделителями "," , "." , " " (запятая, точка, пробел). Признаком конца ввода строки с клавиатуры является нажатие клавиши <Enter>. Длина строки не должна превышать строки экрана (80 символов).

2.1.3. Слово входной строки может состоять из букв (русского и латинского алфавита) и/или цифр.

2.1.4. Первому слову в строке может предшествовать последовательность пробелов.

2.1.5. Признаком конца слова является любой символ ограничитель (см. 2.1.2). Последнее слово строки может не иметь за собой ограничителя, если его последний символ есть последний символ строки.

2.1.6. Ввод пустой строки является указанием пользователя на необходимость закончить работу про-граммы.

2.2. Выходные данные программы.

2.2.1 Сообщения программы.

Сообщения программы должны выводиться с новой строки. Курсор после вывода устанавливается на новую строку.

2.2.1.1. "Введите строку для кодирования" - приглашающее сообщение программы. Указывает на необходимость ввода с клавиатуры входной строки.

2.2.1.2. "Работа закончена" - завершающее сообщение программы. Должно выводиться после ввода пустой строки. Работа программы заканчивается.

2.2.1.3. "Ошибка во входной строке" - информационное сообщение программы. Должно выводиться после ввода входной строки, не соответствующей пункту 2.1. Работа программы должна быть продолжена приглашением ввода новой строки.

2.2.2. Выходная строка - результат кодирования.

2.2.2.1. Выходная строка представляет собой последовательность преобразованных слов, организованную (в смысле разделителей и ограничителей) идентично входной строке.

2.2.2.2. Преобразованные слова выходной строки должны получаться из соответствующих слов входной строки применением к ним процедуры кодирования.

2.2.2.3. При выводе на экран выходная строка должна располагаться строго под входной (разделитель под разделителем, слово под словом).

Рабочая программа дисциплины составлена в соответствии с ОС НИЯУ МИФИ (ФГОС) и учебным планом основной образовательной программы (программ).

Автор(ы):

|  |  |  |
| --- | --- | --- |
|  | Шапкин Павел Александрович, к.т.н. |  |