Курс C++ с нуля. Selfedu

Урок 1. Общие понятия

// комментарии
/* */
int main() { } - точка входа в программу
return 0; - возврат кода завершения.

Переменная:
тип имя;
short (2б) - 32К/65К
int (4б) - 2 150К/4300К
float (4б)
double (8б)
char (1б) - 127/255
bool (1б)

Имя не должно начинаться с цифр
по умолчанию переменные знаковые.

int a;
float b=8,7;
a=b;
a = (int)b // приведение.

int a=10; // Ошибок нет
short b;
b=a; // если int влезет в short.

Урок 2. Арифметические операции

double a b;
cin >> a >> b;
double p = 2*(a+b);

int a=5. b=2 // дробная часть
double c= a/b; отбросится.
// нужно приводить либо добавить .0

Инкремент/декремент.
Var++ быстрее, чем Var=Var+1

a = c++; // сначала присвоение, потом увеличение
b = ++c; // сначала увеличение, потом присвоение.

Остаток от деления:
a = 10 % 3 (=1)
// в 10 содержится 3 раза по 3(9), остается 1)

Варианты присвоения:
+= -= *=, /=, %= - аналожно a = a + d
приоритет:
++/-- */ %, +/-

ДЗ:
float h, d;
cin >> h >> d;
double S = 1/2 * h * d;

float S =
= (1/2) - (1/4) + (1/5) - (1/7) + (1/10);

Урок 3. Директивы препроцессора

#include "" - поиск включаемого файла в тек. каталоге
#include <> - поиск в библиотеках.

#define <id> <текст
(параметры)

#define PX cout << "X = " << X << endl
#define SQUARE(X) X * X
X = SQUARE(3); // произойдет подстановка
PX;

#define P(A, B) 2*(A+B) - макрос.
#undef P - отменяет макрос

#if, #ifdef, #ifndef, #elif, #else, #endif
'условия компиляции'

#ifdef Linux
#else ... // компиляция
#endif ... под разные ОС.

#ifndef // защита от
#define повторного include
#endif

ДЗ:
#define SQUARE
#ifdef SQUARE
A * B;
#else
2 * (A + B);
#endif

Урок 4. Условные операторы

if (условие) оператор
else if (условие) оператор
else оператор

if (x>0) cout <<
else if (x<0) {cout <<
cout <<}
else cout <<...;

Логические операторы: && и, || или, ! не
составные условия: ((x<0 || x>2) && y>=5 && y<=7)

Оператор множественного выбора switch

switch (item) {
case X: оператор;
break;
case X: ...
default: }

Работает быстрее if

ДЗ: if (a<0) a*=-1;
if (b<0) b*=-1;
S=a+b;
if ((a+b)!=b) T=1/(a+b);

Урок 5. Операторы циклов

while (условие) {
тело цикла
}

Хитрый подсчет:
while ((S+=++i)<50);

for (начало; условие; шаг)
{ тело }

можно писать так:

int X=0;
for(;;) {
if (X>1) break;
X+=0,1;}

do {} while (условие); // Сначала делает тело цикла,
// потом проверяет условие

Вложенный цикл:

```
for(i=0; i>10; i++)
    for(j=0; j>10; j++)
        S=i+j;
```

break - немедленное прекращение цикла.
continue - переход на следующую итерацию.

Д3: $S=\prod_{\substack{i=-3\\ i\neq 0}}^{10} i^2$

```
for(i=-3; i<=10; i++)
{ if (i==0) continue;
  S*=i*i;
}
```

```
do{
  in>>a;
  if(a<b) b=a;
  i++;
}while(i<10);
```

Урок 6. Массивы

Тип имя[число элементов]
```
int powers[4]={1,2,3,4};
```
или
```
int powers[]={1,2,3,4};
```

Число элементов
```
int N=sizeof(data)/sizeof(int);
```

Двумерный массив:
```
int a20[3][2]={{1,2},{3,4},{5,6}};
int a20[][]=...
```

Вставка элемента:
```
for (i=8; i>1; --i)
    a[i]=a[i-1];
a[3]=4;
```

Удаление:
```
for(i=5; i<8; ++i)
    a[i]=a[i+1]
```

Сортировка:
через циклы и свап
```
t=a[i];
a[i]=a[pos];
a[pos]=t;
```

Дз: $f(x) = 5x^2 + 6x - 3$; $X \in [-4; 4; 0,1]$

```
for (float i=-4; i<4; i+=0,1)
    cnt++;
for (float X=-4; X<4; X+=0,1)
  { array[X]=5*X*X+6*X-3;
    cnt++; }
```

Урок 7 Строки

Обычные строки:

```
char str[100] = {'П','р','и','в','е','т','\0'};
char str_2[100]="Привет";
char str_3[]="Привет";
либо  strcpy(str,"привет")
```

Функции:

strcat(a "мир!");
strcmp(a,b); - возвращает 0/1;
gets(b); - ввод строки с пробелами
cin>> - вводит до пробела.

```
sprintf(str, "%s..", name); //Форматирует строку по шаблону.
```

Преобразование строк в числа
atoi(), atof().
puts() - вывод строки с пробелами;

Урок 8. Функции

Тип имя(параметры) {тело}

Void - означает пустоту (отсутствует тип)
Return - возврат значения
если Void, то return не нужен.

```
Void print(int a[], int n){ cout<< a[i]<< endl;}
void print(int, int); - прототип
```

прототип нужен для удобства и модульности

Перегрузка:

int modul(int X);
float modul(float X);

Аргументы по умолчанию:
void show(int a=10, int b=20);
show(); - по умолчанию
show(1,2); - переопределение

У функции есть сек вызовов - рекурсия.
void up_and_down(int n){ n=1 n=1 ←
{ if (n<4) up_and_down(n+1); } ↳ n=2 ←
↳ n=3 ←
↳ n=4
конец вызовов.
int a[5] = {1, 2, 3, 4, 5};
int sum(int a[], int n, int sumA){
if (n>1)
{ return a[n-1] + sum(a, n-1, a[n-2]); }
}

УРОК 9. Область видимости.

{} - блок кода / видимости.
при выходе из него - все что внутри уничтожается.
for(int i) - создает свою область для i.
Переменные, объявленные вне {} считаются Глобальными
extern short y; - прототип гл. переменной.
int main()
int x = 3;
extern short y;
cout << ::x; ← обращаемся к глоб. перем.
return 0; }
short y = 5;
Имеет объявление

Глоб. перем. следует избегать.

Статическая переменная: static int...
Создается только один раз в программе.
const переменные нельзя изменять.

Урок 10. Битовые операции

char a; - 1 байт = 8 бит 

X	НЕ
0	1
1	0

В Си++ '~' unsigned char VAR = 153;
unsigned char V = ~VAR (=102)

И - даст 1 если оба 1 unsigned char res = 4 & 5; (=4)
"умножение" Включает и выключает биты

Или - даст 1 если хотя бы 1 есть. res = 8 | 5; (=13)
"сложение" Добавляет биты

XOR - даст 1, если оба разные res = 9 ^ 1; (=8)
Переключение бит.
Применяется в шифровании

Сдвиг вправо: >> "деление на два нацело"
Аналогично сдвиг влево

Урок 11. Структуры

struct имя { переменные };
struct tag book;
тип данных - перемен.

Обращение к полям: структура.поле.

Массив структур: struct book lib[100];
lib[0]. book;

Вложенные структуры: struct people{ people. fio. name;
struct fio;
int old;};

Передача в функцию:
void show(struct people p); p[] - для массива.

struct var { static char a;}

VAR V.a=1 — ссылается на одну и ту же
VAR V1.a=2 статическую переменную.

Урок 12. Объединения

union имя {переменные};
В отличие от структур, union занимает меньше памяти, но использует только одно поле из нескольких
если имя {переменные};
содержит константы 0, 1, 2.
или те, которые зададим сами.

Пользовательские типы:

typedef имя_типа_без. имя;
typedef unsigned char BYTE.
typedef struct VAR VAR;

Урок 13. Память

Указатели:

int a=0;
int *p; - объявим указатель
p=&a; - присвоили адрес.
*p=2; (a=2) - присвоение значения через указ.
int a[]={1,2,3}; - массив, тоже указатель
*p=a;
p++; - сдвиг на int байт (чтобы попасть в a[1])

Разнотипные указатели можно присвоить друг к другу только с преобразованием.

int *p1=&a.
Char* p2=(char*)p1;
// можно перебрать 4 байта int побайтно.

Работа со структурами. VAR V={'#', 5};

Var *p = &v.
Обращение: (*p).a; (*p).b;
либо p->a; p->b;

Выделение памяти: new имя типа
int *p = new int; // выделили 4 байта
*p = 1; // внесли значение
delete p; // освободили память.

Ссылки: int var; // у var два имени:
int &var2 = var; var и var2.
Удобно передавать в функции
копии переменных не создаются.

УРОК 14. Работа с файлами

FILE* fopen(путь, режим);
Возвращает NULL, если файл не открыть
D:/path/file.txt - полный путь
file.txt - файл из текущего каталога
../path/file - файл из директории в родит. катал

Режимы: r, rb - чтение
w, wb - запись с нуля
a, ab - запись к имеющимся

Считывание символа: getc(FILE *fp);
Запись символа: putc(ch, FILE *fp);
fclose(fp); - закрытие файла.

Запись строки: fputs(str, fp);
считывание: fgets(buff, sizeof(buff), fp);

Запись и считывание по шаблону:
fprintf(fp, "%s %s %d" b.author, b.title, b.year);
fscanf(fp, "%s %s %d", b.author, sizeof(b.author), ..., &b.year);

fseek(fp, 0, SEEK_END); // установка позиции
ftell(fp); // возврат текущей позиции.

В бинарном виде:
fwrite((char*)&a, sizeof(int), 1, fp);
fread((char*)&b, sizeof(int), 1, fp);

rename(path, path1); //переименование
remove(path); // удаление

УРОК 15. СТЕК

Стопка объектов с данными:

```
typedef struct{
    char name[50];   // данные элемента
    short old;       // стека.
} Data;

typedef struct element{
    Data d;          // данные
    struct element* next;  // указатель на след.
} OBJ;                     // элемент
```

Функции:

```
OBJ* push(DATA d){                  // добавление
OBJ* PtR = new OBJ;                 // элемента
  PtR->d = d;
  PtR->next = top;
  top = PtR;
  Return PtR;}

void pop(){                         // удаление элемента.
  OBJ* ptr = top->next;
  delete top;
  top = ptr;}

void show(){                        // отображение
  OBJ* c = top;                     // элементов.
  while(c != Null){ cout << ..
                    c = c->next;}}
```

Стандартный стек. #include <stack>

```
stack<DATA> s;
s.push(d);
s.size();
s.top();
s.pop();
```

УРОК 16. Список

Структура динамическая, двунаправленная
В отличие от стека, здесь используется два указателя на начало и конец
Стандартный список:

```
#include<list>
list<Book> mylist;

mylist.push_back(book); // добавление в
mylist.push_front(book); // начало и конец
mylist.begin() mylist.end(); // ук. на начало конец
                             // текущее итератор.
mylist.insert(++iterator, book); // добавление
                                 // в список.
```

УРОК 17. Дерево

см. Структуры данных.

УРОК 18. Указатель на функцию

```
тип (*указатель)(параметры);
Void (*ptr_func)();
PTR_func = foo; // здесь можно подставить разные
PTR_func();     // а вызывать через один указатель.
```

Пример: void show_info const char* (*info)(); info();

УРОК 19. Автоматические переменные

auto a = 5;
тип определяется на этапе компиляции.

Цикл по коллекциям:
for (auto val : ar) {}
val - копия коллекции
Для изменения элементов используем
for (auto& val : ar)

ООП
Урок 1. Общие понятия
Наследование - Создание классов на основе других
Полиморфизм - переопределение родительских функций
Инкапсуляция - скрытие данных внутри класса

class имя
{
данные и методы;
};
Point pt;
Point * pt_ptr = new Point();
delete pt_ptr;
Private - закрытая область класса.
Public - открытая область класса.
Принято разделять объявление и реализацию:

point.h
#define POINT
class Point
{ private:
int x,y;
Public:
void setCoord(int x, int g);

point.cpp
#include "point.h"
Void Point::setCoord(x,g)
{ }

int main()
Point* pt=new Point();
pt→setCoord(1,2);
delete pt;

Урок 2. Set и Get.

set-метод - устанавливает значение закрытой переменной
get-метод - возвращает закрытую переменную.
Оба метода можно делать проверками.
Скрытый указатель this, указывает на конкретный объект класса.
Можно явно указать, перегружую какой класса использовать.
Объекты можно передавать и возвращать из функции,
но лучше делать по ссылке

```
Point& foo(Point& a)
{
return a;
}
```

УРОК 3. Конструктор/деструктор

Создают и удаляют объекты класса
Поумолчанию скрыты, но можно задать вручную

```
class Point {
public:
    Point(); // конструктор по умолчанию
    ~Point(); // деструктор
};

Point(int x) {} // перегрузка конструктора
Point(int x, int y): Point(x) {} // делигирующий констр.
```

Инициализация через конструктор:

```
Point(int px, int py): x(px), y(py) {}
```

УРОК 4. Конструктор копирования

Скопировать объект можно через функцию:

```
Point foo(Point pt)
{ здесь будет копия
```

Либо так: Point a;
Point b(a);
Происходит копирование одной области памяти в другую.
При использовании конструктора копирования.
Point(const Point& other)
{ this->x = other.x;
this->y = other.y; }

Урок 5. Статические переменные

Единая переменная для всех объектов
Не размножается по объектам класса, а
существует внутри класса.
static int Count;
обратиться можно так: int Point::nCount=0
либо через объект класса: pt.Count
Это для public переменных
Для private переменной нужен статический Get-
Point::GetCount() // так же только один.
pt.GetCount
Через статику можно не создавая объекты
вызывать функции.

УРОК 6. Агрегация и делегирование

Агрегация + Отношение "часть-целое"
То есть часть ~~вложена в целое или~~
передана по ссылке и существует пока
class Dog { — пример композиции. — сильно
public:
void run() { feet.Run(); }
private:
class Feet { void run() {}; } ← вложенный класс
Feet feet; }; и его объект.

Композиция – когда часть существует только внутри целого.
"Пример описанный ранее на класс Feet нужно вынести наружу"

Урок 7. Наследование

Базовый класс
↓ - наследование.
Производный класс
со свойствами базового
+ свои собственные.

```
class Point {
    Point() {X=0; Y=0;};
class Properties{
    Properties();
  Protected:
    Point sp, ep;
    short width;};
class Line: public Properties{
    Line();
```

наследование.

При создании объекта вызывается конструктор базового класса, а потом деструктор.

Явный вызов конструктора базового класса:
Line(): Properties(X1, Y1, X2, Y2)

Наследуются не только данные, но и методы
Цепочка наследования неограничена, но могут возникнуть проблемы со множественным наследованием.

Урок 8. Перегрузка методов и виртуальные функции

Перегрузка:

```
class Prop{
    void Draw();
};
```

```
Class Line: pubic Prop{
    void Draw();
};
```

Line line;
line.Draw(); - вызовет метод класса.

Line line;
Rect* pr = &line; - указатель на базовый класс Line
pr->Draw(); - вызовется род. метод.

То есть мы переопределили метод.
чтобы метод дочернего вызывался из базового,
нужно его сделать виртуальным.

virtual void Draw()
Line line; ссылка на доч. класс
line.Draw();
void Draw() override {}
Переопределяется самый последний метод.

Prop* obj[2];
obj[0] = new Line();
obj[1] = new Rect();
for(auto& o: obj)
o->Draw();

Работа с разными объектами через базовый класс

Урок 9. Дружественные классы.

Разрешает доступ к приватным данным класса.

class Line; Rect;
class Prop{
private:
public:
friend Line; Rect; - др. класс
friend void showInfo(); - др. функция

Урок 10. Плюс. наследование

class Line: public Styles, public Pos{

Что делать если в под. классах есть одинаковый метод
POS *'ps = &line'
Вызывает нагружную через указатель
ps → getInfo();

Ромбовидное наследование?
class.. : public virtual Graph{}

УРОК 11. Виртуальные классы

class Weapon { — абстрактный класс
public: нельзя создать объект.
virtual void Shoot()=0;};
class Gun : public Weapon { — перегрузка.
void Shoot() override {};

УРОК 12. Перегрузка оператора класса

Чтобы не было ошибок double free при копирова-
нии, нужно перегрузить оператор =

```
Void operator= (const Point & other) {
    delete[] coords;
    this → size = other.size;
    this → coords = new int[this → size];
    for(...)
```

Для цепной перегрузки дописать:
void operator +
const Point & operator +

Для копирования при создании нужен констр.
копирования (аналогичен перегрузке)
Point P1 = P2;

УРОК 13. Блоки try catch

ifstream ifs;

ifs- генерирует исключение если будет ошибка.
try{
ifs.open(...)
}
catch (istream::failure& e) - обрабатывает ошибку
e.what()
Генерация исключений:
if(b==0) throw 0;
catch (int& e)
Для каждого исключения нужен свой блок catch.
Класс исключение.
class Exception: public exception (std)
public:
Exception(char* error): exception(error)
{this->error= error;}
private: string error
throw Exception(...)
catch(Exception& e);

Урок 14. Шаблоны функций и классов

template<typename T> - объявляем шаблон функции
T pow(T a, T b){}
можно задать несколько типов через запятую
template<class T> - шаблон класса
class Point{
Point(T x, T y){}};
Point<int> pt(5,6); - для создания объекта нужен тип.
Специализация шаблона:
template<>
class Point<string>
...
Спец. шаблон для работы со строками.
От шаблонов можно наследоваться:
class Point3D: public Point<T>

УРОК 16. Пространства имен

Обращение к глобальной области - ::

Задаем свое пространство:

namespace ... {}
namespace ... {}
namespace ... {
namespace ... {}
}

можно писать несколько раз.
можно вкладывать

Использование пространства: using namespace...
обращение: std::cout.

УРОК 17. Класс-обёртка и лямбда

function<void()> f; - <тип возврата(тип арг)>
f = foo; - инициализация
f(); = вызов.

Обертка над функцией foo.

[](int a){} - анонимная функция
Вызвать можно через обертку.
function<void(int)> f;
f = [](int a){}
Её можно передать другой функции
doWork(cal, size, [](int a) - вызов функции
{ }; - дополнительная реализация.
аргументы передаются в [] без типа.
auto f = [&size](){ return size; }
Тип возврата автоматически задается.

УРОК 19. Умные указатели

```
template<class T>
class SmartPointer{
public: SmartPointer(T *ptr){
            this->ptr = ptr;}
```

~SmartPointer() { delete ptr; }
T& operator*() override { return *ptr; }
private: T* ptr; };
SmartPointer<int> p = new int(5);

Стандартные указатели.
auto_ptr<int> p(new int(5)); - устаревший
auto_ptr<int> p2 = p; - не копирует память.
unique_ptr<int> p(new int(5));
p2 = move(p) - копирует память.
shared_ptr - указывает на одну память и
два раза не удаляет указатель.