C++

0. Основы C++

0.1. История создания

Года создания: 1980-е.
Автор: Бьерн Страуструп.
Название предложил: Рик Масситти.

0.2. Парадигмы

Процедурное программирование.
Объектно-ориентированное программирование.
Обобщенное программирование.

0.3. Структура программы C++

<Директивы препроцессора>
<Описание типов пользователя>
<Прототипы функций>
<Описание глобальных переменных>
<Тела функций>

0.4. Препроцессор

Препроцессор — часть компилятора, предназначенная для предварительной обработки текста программы.
Директива – указание препроцессору. Начинаются с символа «#».
Основные директивы препроцессора:

- *#include* — вставляет текст из указанного файла
- *#define* — задаёт макроопределение (макрос) или символическую константу
- *#undef* — отменяет предыдущее определение
- *#if* — осуществляет условную компиляцию при истинности константного выражения
- *#ifdef* — осуществляет условную компиляцию при определённости символической константы

0.5. Потоковый ввод-вывод в C++

В C++ присутствует библиотека ввода-вывода **#include <iostream>.**
Функции ввода-вывода:

- cin стандартный входной поток (stdin в C)
- cout стандартный выходной поток (stdout в C)
- cerr стандартный поток вывода сообщений об ошибках (stderr в C)

Для их использования в Microsoft Visual Studio необходимо прописать строку: **using namespace std;**

0.6. Форматированный ввод-вывод в C++

Таблица флагов форматирования

Флаг	Назначение	Пример	Результат
boolalpha	Вывод логических величин в текстовом виде (`true` , `false`)	**cout.setf(ios::boolalpha); bool log_false = 0, log_true = 1; cout << log_false << endl << log_true << endl;**	**false true**
oct	Ввод/вывод величин в восьмеричной системе счисления (сначала снимаем флаг `dec` , затем устанавливаем флаг `oct`)	**cout.unsetf(ios::dec); cout.setf(ios::oct); int value; cin >> value; cout << value << endl;**	ввод:99_{10} вывод:143_{8}
dec	Ввод/вывод величин в десятичной системе счисления (флаг установлен по умолчанию)	**cout.setf(ios::dec); int value = 148; cout << value << endl;**	**148**
hex	Ввод/вывод величин в шестнадцатеричной системе счисления (сначала снимаем флаг `dec` , затем устанавливаем флаг `hex`)	**cout.unsetf(ios::dec); cout.setf(ios::hex); int value; cin >> value; cout << value << endl;**	ввод:99_{10} вывод:63_{16}
showbase	Выводить индикатор основания системы счисления	**cout.unsetf(ios::dec); cout.setf(ios::oct | ios::showbase); int value; cin >> value; cout << value << endl;**	ввод:99_{10} вывод:0143_{8}
uppercase	В шестнадцатеричной системе счисления использовать буквы верхнего регистра(по умолчанию установлены буквы нижнего регистра)	**cout.unsetf(ios::dec); cout.setf(ios::hex | ios::uppercase); int value; cin >> value; cout << value << endl;**	ввод:255_{10} вывод:FF_{16}

showpos	Вывод знака плюс `+` для положительных чисел	**cout.setf(ios::showpos); int value = 15; cout << value << endl;**	**+15**
scientific	Вывод чисел с плавающей точкой в экспоненциальной форме	**cout.setf(ios::scientific); double value = 1024.165; cout << value << endl;**	**1.024165e+003**
fixed	Вывод чисел с плавающей точкой в фиксированной форме(по умолчанию)	**double value = 1024.165; cout << value << endl;**	**1024.165**
right	Выравнивание по правой границе(по умолчанию). Сначала необходимо установить ширину поля(ширина поля должна быть заведомо большей чем, длина выводимой строки).	**cout.width(40); cout << «cppstudio.com» << endl;**	**__cppstudio.com**
left	Выравнивание по левой границе. Сначала необходимо установить ширину поля(ширина поля должна быть заведомо большей чем, длина выводимой строки).	**cout.setf(ios::left); cout.width(40); cout << «cppstudio.com» << endl;**	**cppstudio.com__**

Таблица манипуляторов форматирования

Манипулятор	Назначение	Пример	Результат
endl	Переход на новую строку при выводе	**cout << «website:» << endl << «cppstudio.com»;**	**website: cppstudio.com**
boolalpha	Вывод логических величин в текстовом виде (`true` , `false`)	**bool log_true = 1; cout << boolalpha << log_true << endl;**	**true**
noboolalpha	Вывод логических величин в числовом виде (`true` , `false`)	**bool log_true = true; cout << noboolalpha << log_true << endl;**	**1**
oct	Вывод величин в восьмеричной системе счисления	**int value = 64; cout << oct << value << endl;**	**100_8**
dec	Вывод величин в десятичной системе счисления (по умолчанию)	**int value = 64; cout << dec << value << endl;**	**64_{10}**
hex	Вывод величин в шестнадцатеричной системе счисления	**int value = 64; cout << hex << value << endl;**	**40_8**
showbase	Выводить индикатор основания системы счисления	**int value = 64; cout << showbase << hex << value << endl;**	**0x40**
noshowbase	Не выводить индикатор основания системы счисления (по умолчанию).	**int value = 64; cout << noshowbase << hex << value << endl;**	**40**

uppercase	В шестнадцатеричной системе счисления использовать буквы верхнего регистра (по умолчанию установлены буквы нижнего регистра).	**int value = 255; cout << uppercase << hex << value << endl;**	**FF_{16}**
nouppercase	В шестнадцатеричной системе счисления использовать буквы нижнего регистра (по умолчанию).	**int value = 255; cout << nouppercase << hex << value << endl;**	**ff_{16}**
showpos	Вывод знака плюс `+` для положительных чисел	**int value = 255; cout << showpos<< value << endl;**	**+255**
noshowpos	Не выводить знак плюс `+` для положительных чисел (по умолчанию).	**int value = 255; cout <<noshowpos<< value << endl;**	**255**
scientific	Вывод чисел с плавающей точкой в экспоненциальной форме	**double value = 1024.165; cout << scientific << value << endl;**	**1.024165e+003**
fixed	Вывод чисел с плавающей точкой в фиксированной форме (по умолчанию).	**double value = 1024.165; cout << fixed << value << endl;**	**1024.165**
setw(int number)	Установить ширину поля, где `number` — количество позиций, символов (выравнивание по умолчанию по правой границе). Манипулятор с параметром.	**cout << setw(40) << «cppstudio.com» << endl;**	**__cppstudio.com**
right	Выравнивание по правой границе(по умолчанию). Сначала необходимо установить ширину поля(ширина поля должна быть заведомо большей чем, длина выводимой строки).	**cout << setw(40) << right << «cppstudio.com» << endl;**	**__cppstudio.com**
left	Выравнивание по левой границе. Сначала необходимо установить ширину поля(ширина поля должна быть заведомо большей чем, длина выводимой строки).	**cout << setw(40) << left << «cppstudio.com» << endl;**	**cppstudio.com__**
setprecision(int count)	Задаёт количество знаков после запятой, где `count` — количество знаков после десятичной точки	**cout << fixed << setprecision(3) << (13.5 / 2) << endl;**	**6.750**
setfill(int symbol)	Установить символ заполнитель. Если ширина поля больше, чем выводимая величина, то свободные места поля будут наполняться символом `symbol` — символ заполнитель	**cout << setfill('0') << setw(4) << 15 << ends << endl;**	**0015**

0.7. Базовые типы данных

- **int** — целочисленный тип данных.
- **float** — тип данных с плавающей запятой.
- **double** — тип данных с плавающей запятой двойной точности.
- **char** — символьный тип данных.
- **bool** — логический тип данных

0.8. Приведение типов данных

Приведение типа — превращение значения одного типа в значение другого типа. При приведении дробного к целому возможна потеря данных.

Неявное преобразование типов данных выполняет компилятор C++.

Явное преобразование данных выполняет сам программист.

Синтаксис преобразования:

- (тип_данных) имя_переменной
- static_cast<тип_данных>(имя_переменной)

0.9. Операция присваивания

В C++ существует пять операций присваивания, не считая основную операцию присваивания: «=»

- **+=** операция присваивания-сложения;
- **-=** операция присваивания-вычитания;
- ***=** операция присваивания-умножения;
- **/=** операция присваивания-деления;
- **%=** операция присваивания-остатка от деления;

0.10. Арифметические и логические операции

Арифметические операции

Основные **бинарные операции**, расположенные в порядке уменьшения приоритета:

- * — умножение;
- / — деление;
- + — сложение;
- — — вычитание;
- % — остаток от целочисленного деления.

Основные **унарные операции:**

- ++ — инкрементирование (увеличение на 1);
- - - — декрементирование (уменьшение на 1);
- — — изменение знака.

Логические операции

Логические операции делятся на две группы:

- условные;
- побитовые.

Основные **условные** логические операции:

- **&&** — логическое И (бинарная);
- || — логическое ИЛИ (бинарная);
- **!** — логическое НЕ (унарная).

Основные **побитовые** логические операции в языке Си:

- **&** - конъюнкция (логическое И);
- **|** - дизъюнкция (логическое ИЛИ);
- **~** - инверсия (логическое НЕ);
- **^** - исключающее ИЛИ;
- **>>** - поразрядный сдвиг вправо;
- **<<** - поразрядный сдвиг влево.

0.11. Приоритет операций

1.	Разрешение области действия	**::**
2.1 **2.2**	Унарные Другие	Префикс ++, --, ->* ( ), [ ], .
3.	Унарные	+, -, !, *&, явное преобразование
4.	Арифметические	*, /, %
5.	Арифметические	+, -
6.	Поразрядный сдвиг	<<, >>
7.	Сравнение	>, <, >=, <=

8.	Сравнение	==, !=
9.	Поразрядные логические	&
10.	Поразрядные логические	^
11.	Поразрядные логические	|
12.	Логические	&&
13.	Логические	||
14.	Условные	?:
15.	Присваивание	=, *=, +=, /=, %=, -=, <<=, >>=, &=
16.1 **16.2**	Унарные Последовательные	++ ,

0.12. Функции библиотеки math.h

Функция	**Описание**
Округление	
`round`	Округляет число по правилам арифметики, то есть `round(1.5) == 2`, `round(-1.5) == -2`
`floor`	Округляет число вниз (“пол”), при этом `floor(1.5) == 1`, `floor(-1.5) == -2`
`ceil`	Округляет число вверх (“потолок”), при этом `ceil(1.5) == 2`, `ceil(-1.5) == -1`
`trunc`	Округление в сторону нуля (отбрасывание дробной части), при этом `trunc(1.5) == 1`, `trunc(-1.5) == -1`
`fabs`	Модуль (абсолютная величина)
Корни, степени, логарифмы	
`sqrt`	Квадратный корень. Использование: `sqrt(x)`
`cbrt`	Кубический корень. Использование: `cbrt(x)`
`pow`	Возведение в степень, возвращает a^b. Использование: `pow(a,b)`
`exp`	Экспонента, возвращает e^x. Использование: `exp(x)`
`log`	Натуральный логарифм
`log10`	Десятичный логарифм
Тригонометрия	
`sin`	Синус угла, задаваемого в радианах
`cos`	Косинус угла, задаваемого в радианах
`tan`	Тангенс угла, задаваемого в радианах

asin	Арксинус, возвращает значение в радианах
acos	Арккосинус, возвращает значение в радианах
atan	Арктангенс, возвращает значение в радианах

0.13. Операторы switch и case of

Switch – оператор множественного выбора.
Case – лейбл, должен иметь уникальное значение.
Break – прерывает выполнение.
Пример программы:

```
switch (p)
  {
    case '0': // если p = 0
    case '1': // если p = 1
    case '2': // если p = 2
    case '3': // если p = 3
    case '4': // если p = 4
    case '5': // если p = 5
    case '6': // если p = 6
    case '7': // если p = 7
    case '8': // если p = 8
    case '9': // если p = 9
      return true; // то возвращаем true
    default: // в противном случае, возвращаем false
      return false;
  }
```

1. Одномерные массивы

1.1. Объявление массива

Понятие массива: **Массив** – набор переменных одинакового типа. Доступ к этим переменным осуществляется по одному имени. Это имя называется именем массива.

тип имя_массива[размер];

тип – это тип элементов массива. Он еще называется базовым типом. Базовый тип определяет количество данных каждого элемента, который

составляет массив. Тип элементов массива может быть как базовым типом так и составным (например, структура).

размер – количество элементов в массиве;

имя_массива – непосредственно имя массива, по которому осуществляется доступ к элементам массива.

1.2. Способы заполнения

Инициализация числовых массивов

Общий вид инициализации с заданием размера массива:
тип имя_массива[размер] = { список_значений };

Общий вид «безразмерной» инициализации:
тип имя_массива[] = { список_значений };

Инициализация символьных массивов

char имя_массива[размер] = "строка";

char имя_массива[] = "строка";

char имя_массива[] = { 'с', 'т', 'р', 'о', 'к', 'а' };

Заполнение с клавиатуры

```
Int A[N];
For (int I = 0; I < N; I++) {
        Cin >> A[i];
}
```

Заполнение случайными числами из промежутка [a, b]

```
Int A[N];
For (int I = 0; I < N; I++) {
        A[I] = rand%((b - a)+1) + a;
}
```

Заполнение случайными действительными числами из промежутка

```
Int A[N];
For (int I = 0; I < N; I++) {
```

```
        A[I] = (double)(rand())/RAND_MAX*(в - а) + а;
}
```

1.3. Поиск в массиве

Поиск элемента X

```
i = 0;
while ( i < N && A[i] != X )
i ++;
if ( i < N )
cout <<"A[" << i << "]="<< X << endl;
else
cout << "Такого нет!" << endl;
```

1.4. Максимальный(минимальный) элемент

Поиск максимального элемента в массиве

```
M = A[0];
for ( i = 1; i < N; i++ )
if ( A[i] > M ) M = A[i];
cout << M << endl;
```

Или

```
M = A[0]; nMax = 0;
for ( i = 1; i < N; i++ )
if ( A[i] > = A[nMax] ) nMax = i;
cout <<"A[" << nMax << "]="<< A[nMax] << endl;
```

1.5. Реверс массива

```
for ( i = 0; i < (N/2); i++ )
{
c = A[i];
A[i] = A[N-1-i];
A[N-1-i] = c;
}
```

1.6. Сдвиг элементов

```
c = A[0];
for ( i = 0; i < N-1; i++ )
A[i] = A[i+1];
A[N-1] = c;
```

1.7. Отбор необходимых элементов (запись в другой массив)

```
count = 0;
for ( i = 0; i < N; i++ )
if ( A[i] % 2 == 0 )
{
B[count] = A[i];
count ++;
}
```

1.8. Двоичный поиск

Примеры двоичного поиска в реальной жизни – поиск слова в словаре, угадывание чисел.

```
int X, L, R, c;
L = 0; R = N;
while ( L < R-1 )
{
c = (L+R) / 2;
if ( X < A[c] )
R = c;
else
L = c;
}
if ( A[L] == X )
cout << "A[" << L << "]= " << X << endl;
else
cout << "Не знайшли!"<< endl;
```

1.9. Сортировка массива

1.9.1. Метод пузырька (сортировка обменом)

Суть:

для j від N-2 до 0 крок -1
якщо A[j+1]< A[j] то
поміняти місцями A[j] и A[j+1]

Пример:

```
//Сортировка по убыванию
For (int i = 1; i < n; ++i)
{
	For (int r = 0; r < n-i; r++)
	{
		If (mass[r] < mass[r+1])
		{
			// Обмен местами
			int temp = mass[r];
			mass[r] = mass[r+1];
			mass[r+1] = temp;
		}
	}
}
```

1.9.2. Метод выбора

Суть:

для i від 1 до N-1
знайти номер nMin мінімального елемента з A[i] .. A[N]
якщо i! = nMin то
поміняти місцями A[i] та A[nMin]

Пример:

```
for ( i = 0; i < N-1; i++ ) {
nMin = i;
```

```
for ( j = i+1; j < N; j++ )
if ( A[j] < A[nMin] )
nMin = j;
if ( i != nMin )
{
// поміняти місцями A[i] и A[nMin]
                    int temp = A[i];
                    A[i] = A[nMin];
                    A[nMin] = temp;
}
}
```

2. Многомерные массивы

2.1. Объявление массива и заполнение элементами

```
int mas [2][5];
int mas [2][5] ={ 1, 5, 3, 7, 4, 10, 11, 13, 14, 25 };
int mas [ ][5] ={ 1, 5, 3, 7, 4, 10, 11, 13, 14, 25 };
int mas [ ][5] ={ { 1, 5, 3, 7, 4 },{10, 11, 13, 14, 25} };

mas[0][0] = 10;
mas[0][1] = 5;
mas[1][0] = 12;

for (I = 0; I < n; i++)
for (j = 0; j <m; j++)
    cin >>mas[i][j];

for (I = 0; I < n; i++)
for (j = 0; j <m; j++)
    mas[i][j] = rand%((b - a)+1) + a;
```

2.2. Обработка матриц (на примере поиска максимального элемента заштрихованной области)

const int q=6;
int A[q][q];

	for(int i=0;i<q;i++)
	{
	for(int j=0;j<q;j++)
	{
	if(max<A[i][j]&&j>=i)
	max=A[i][j];
	}
	cout<<"\n\n";
	}

	for(int i=0;i<q;i++)
	{
	for(int j=0;j<q;j++)
	{
	if(max<A[i][j]&&j!=i+1)
	max=A[i][j];
	}
	cout<<"\n\n";
	}

	for(int i=0;i<q;i++)
	{
	for(int j=0;j<q;j++)
	{
	if(max<A[i][j]&&j>=i&&j<=q-i-1)
	max=A[i][j];
	}
	cout<<"\n\n";
	}

	for(int i=0;i<q;i++)
	{
	for(int j=0;j<q;j++)
	{
	if(max<A[i][j]&&j<=i&&j>=q-i-1)
	max=A[i][j];
	}
	cout<<"\n\n";
	}

	for(int i=0;i<q;i++)
	{
	for(int j=0;j<q;j++)
	{
	if(max<A[i][j]&&j<=i&&j>=q-i-
	1||j>=i&&j<=q-i-1&&max<A[i][j])
	max=A[i][j];

	} cout<<"\n\n"; }

	for(int i=0;i<q;i++)
	{
	for(int j=0;j<q;j++)
	{
	if(max<A[i][j]&&j>=i&&j>=q-i-1||j<=i&&j<=q-i-1&&max<A[i][j])
	max=A[i][j];
	}
	cout<<"\n\n";
	}

	for(int i=0;i<q;i++)
	{
	for(int j=0;j<q;j++)
	{
	if(max<A[i][j]&&j<=i&&j<=q-i-1)
	max=A[i][j];
	}
	cout<<"\n\n";
	}

	for(int i=0;i<q;i++)
	{
	for(int j=0;j<q;j++)
	{
	if(max<A[i][j]&&j>=i&&j>=q-i-1)
	max=A[i][j];
	}
	cout<<"\n\n";
	}

	for(int i=0;i<q;i++)
	{
	for(int j=0;j<q;j++)
	{
	if(max<A[i][j]&&j<=q-i-1)
	max=A[i][j];
	}
	cout<<"\n\n";
	}

	for(int i=0;i<q;i++)
	{
	for(int j=0;j<q;j++)
	{
	if(max<A[i][j]&&j>=q-i-1)
	max=A[i][j];
	}
	cout<<"\n\n";
	}

2.3. Поиск в строке/столбце матрицы

Поиск максимального элемента в каждой строке матрицы:

```
for(i=0;i<n;i++)
   {
      max=a[i][0];
      for (int j=1; j<m; j++)
      {
         if ( a[i][j]>max)
         {
            max=a[i][j];
         }
      }
      cout << "Max of " << i << " line=" << max << endl;
   }
```

Поиск максимального элемента в каждом столбце матрицы:

```
for (j=0;j<=m;i++)
{
        max=a[0,j];
        for (i=1;i<=n;i++)
                if (max<a[i,j])
        b[j]=max;
}
```

3. Функции

3.1. Область видимости переменных

Блок (также говорят блок кода, блок команд, блок инструкций) в программировании — это логически сгруппированный набор идущих подряд инструкций в исходном коде программы. Блоки служат для ограничения области видимости переменных и функций

Область видимости переменных — части программы, в которой пользователь может изменять или использовать переменные.

Глобальными переменными - называются те переменные, которые были созданы вне тела какого-то блока. Их можно всегда использовать во всей вашей программе, вплоть до ее окончания работы.

Локальные переменные — это переменные созданные в блоках. Областью видимости таких переменных является блоки (и все их дочерние)

Глобальная переменная уступает локальной

Глобальный оператор разрешения — это два подряд поставленные двоеточия, с помощью которых мы говорим компилятору, что хотим использовать глобальную переменную, а не локальную.

3.2. Объявление функции

Функция определяет действия, которые выполняет программа. Функции позволяют выделить набор инструкций и придать ему имя. А затем многократно по присвоенному имени вызывать в различных частях программы. По сути функция - это именованный блок кода.

Определение функции:

```
тип имя_функции(параметры)
{
   инструкции
}
```

Перед **вызовом** функции надо ее дополнительно объявить:

```
Int func_name ();

Int main (){

        …
        func_name();
}

int func_name () {
        …;
}
```

3.3. Типы функций

Тип функции - тип возвращаемого ней значения: int, double, …

Функции типа void значение не возвращают и называются **процедурами**

3.4. Встроенные функции

Inline-функции обрабатываются точно также как и макрос. При вызове такой функции с вызывающего кода, тело функции непосредственно вставляется в этот код. Иначе говоря, код inline-функции подставляется в то место строки программы, из которого она вызывается. В результате, вызов inline-функции дает выигрыш во времени выполнения программы (времени обработки функции). Это связано с тем, что исчезают накладные расходы на дополнительную обработку при передаче (получении) параметров в функцию.

3.5. Способы передачи аргументов в функцию

Аргументы могут передаваться **по значению** (by value) и **по ссылке** (by reference).

При передаче аргументов по значению внешний объект, который передается в качестве аргумента в функцию, не может быть изменен в этой функции.

void square(int a, int b)

При передаче параметров по ссылке передается ссылка на объект, через которую мы можем манипулировать самим объектов, а не просто его значением. Передача по ссылке позволяет возвратить из функции сразу несколько значений. Также передача параметров по ссылке является более эффективной при передаче очень больших объектов. Поскольку в этом случае не происходит копирования значений, а функция использует сам объект, а не его значение.

void square(int &a, int &b)

3.6. Передача массивов в функцию

Когда массив используется в качестве аргумента функции, передается только адрес массива, а не копия всего массива.

void display(int num[10])

void display(int num[])

Передача по указателю:

*void display(int *num)*

Передача динамического двумерного массива:

*void func(int **Arr)*

3.7. Перегрузка функции

Перегрузка функций — определение несколько функций с одним и тем же именем, но с разными параметрами.

Перегрузку можно выполнять:

- По типу параметров:

  int subtract (int a, int b)
  double subtractDouble(double a, double b)

- По количеству параметров

  int subtract (int a, int b)
  int subtract(int a, int b, int c)

3.8. Рекурсивные функции

Рекурсия – вызов функцией самой себя

Пример – вычисление факториала:

```
int factorial(int n)
{
   if(n>1)
      return n * factorial(n-1);
   return 1;
}
```

Важным отличием циклов от рекурсивных функций является тот факт, что циклы используют для подсчета числа исполнений итератор, а рекурсивные функции для определения момента выхода должны выполнять сравнение результатов.

3.9. Классы памяти

Класс памяти переменной - понятие в некоторых языках программирования. Он определяет область видимости переменной, а также как долго переменная находится в памяти.

- **auto** — автоматическая (локальная). Автоматические переменные создаются при входе в функцию и уничтожаются при выходе из неё
- **static** — статическая переменная (локальная). 1) Если static - внутри функции. Для таких переменных область видимости обычная (внутри функции), но время жизни постоянное (значение сохраняется между вызовами функции). 2) static вне функции имеет другое значение.
- **extern** — внешняя (глобальная) переменная. Внешние переменные доступны везде, где описаны, а не только там, где определены.
- **register** — регистровая переменная (локальная). Это слово является всего лишь «рекомендацией» компилятору помещать часто используемую переменную в регистры процессора для ускорения программы.

4. Работа с файлами

4.1. Основы

Файл – именованный набор байтов, который может быть сохранен на некотором накопителе.

Для работы с файлами необходимо подключить заголовочный файл **<fstream>.**

В <fstream> определены несколько классов и подключены заголовочные файлы **<ifstream>** — файловый ввод и **<ofstream>** — файловый вывод.

4.2. Этапы работы

1) создать объект класса ofstream;
2) связать объект класса с файлом, в который будет производиться запись;
3) записать строку в файл;
4) закрыть файл.

4.3. Пример работы

Открытие файла:

```
ifstream obj_name;
obj_name.open("Filename_1.txt");
```

Считывание из файла в массив:

```
while (!obj_name.eof())
{
obj_name >> A[i];
i++;
}
```

Запись из массива в файл:

```
ofstream ob_name;
ob_name.open("Filename_2.txt");
for (I = 0; I < N; i++)
ob_name << A[i];
```

Закрытие файла:

```
obj_name.close();
```

5. Обработка исключений

5.1. Суть

Исключения в C++ реализованы с помощью трёх ключевых слов, которые работают в связке друг с другом: **throw, try** и **catch.**

Оператор **throw** используется, чтобы сигнализировать о возникновении исключения или ошибки

Блок **try** действует как наблюдатель, в поисках исключений, которые были выброшены каким-либо из операторов в этом же блоке try.

Ключевое слово **catch** используется для определения блока кода (так называемого «блока catch»), который обрабатывает исключения определённого типа данных.

5.2. Пример

```
try
   {
      // Здесь мы пишем код, который может вызвать исключение
      throw 777;
   }
   catch (int a)
   {
      // Любые исключения типа int, сгенерированные в блоке try выше, обрабатываются здесь
      std::cerr << "Ошибка №" << a << '\n';
   }
```

Вывод программы:

Ошибка № 777

6. Структуры

6.1. Понятие структуры

Структура – пользовательский тип данных. Структура позволяет сгруппировать переменные разных типов в единое целое.

6.2. Объявление структуры

```
struct Human //объявление структуры
{
   int age;
   double height; //поля структуры
};
```

Чтобы **использовать** структуру *Human*, нам нужно просто объявить переменную типа *Human:*

```
Human your_name;
```

6.3. Доступ к членам структуры

Для того, чтобы получить доступ к отдельным её членам, используется **оператор выбора члена** «.»

```
Human John; // создаём отдельную структуру Human для John-а
John.age = 27; // присваиваем значение члену age структуры John
```

Есть более быстрый способ инициализации структур с помощью **списка инициализаторов**. Он позволяет инициализировать некоторые или все члены структуры во время объявления переменной типа struct:

```
Human john = { 27, 180}; //  john.age = 27, john.height = 180
```

6.4. Передача структуры в функцию

```
void printInformation(Human human)
{
   std::cout << "Age: " << human.age << "\n";
```

```
    std::cout << "Height: " << human.height << "\n";
}

int main {
        printInformation(john);
}
```

Функция также может возвращать структуру (это один из тех немногих случаев, когда функция может возвращать несколько переменных).

7. Работа со строками

7.1. Основы

В языке C++ для удобной работы со строками есть класс **string**, для использования которого необходимо подключить заголовочный файл string.

7.2. Конструкторы строк

- *string()* - конструктор по умолчанию (без параметров) создает пустую строку.
- *string(string & S)* - копия строки S
- *string(size_t n, char c)* - повторение символа с заданное число n раз.
- *string(size_t c)* - строка из одного символа c.
- *string(string & S, size_t start, size_t len)* - строка, содержащая не более, чем len символов данной строки S, начиная с символа номер start.

7.3. Ввод – вывод строк

Строка выводится точно так же, как и числовые значения:

cout << S;

Для считывания строки можно использовать операцию ">>" для объекта cin:

cin >> S;

Можно считывать строки до появления символа конца строки при помощи функции getline. Сам символ конца строки считывается из входного потока, но к строке не добавляется:

getline(cin S);

7.4. Арифметические операторы

- **=** - присваивание значения.
- **+=** - добавление в конец строки другой строки или символа.
- **+** - конкатенация двух строк, конкатенация строки и символа.
- **==, !=** - посимвольное сравнение.
- **<, >, <=, >=** - лексикографическое сравнение.

7.5. Методы строк

- **Size()** - возвращает длину строки
- **Resize(n)** - Изменяет длину строки, новая длина строки становится равна n. При этом строка может как уменьшится, так и увеличиться
- **Clear()** - очищает строчку, строка становится пустой.
- **Empty()** - возвращает true, если строка пуста, false - если непуста.
- **push_back(c)** - добавляет в конец строки символ c, вызывается с одним параметром типа char.
- **Append()** - Добавляет в конец строки несколько символов, другую строку или фрагмент другой строки. Имеет много способов вызова.
- **Erase()** – удаляет символы из строки.
- **Insert()** - Вставляет в середину строки несколько символов, другую строку или фрагмент другой строки.
- **Substr()** - возвращает подстроку данной строки.
- **Replace()** - Заменяет фрагмент строки на несколько равных символов, другую строку или фрагмент другой строки
- **Find()** - Ищет в данной строке первое вхождение другой строки
- **Rfind()** - Ищет последнее вхождение подстроки ("правый" поиск). Способы вызова аналогичны способам вызова метода find.
- **find_first_of()** - Ищет в данной строке первое появление любого из символов данной строки str. Возвращается номер этого символа или значение string::npos.
- **find_last_of()** - Ищет в данной строке последнее появление любого из символов данной строки str. Способы вызова и возвращаемое значение аналогичны методу find_first_of.

- **c_str()** - Возвращает указать на область памяти, в которой хранятся символы строки, возвращает значение типа char*. Возвращаемое значение можно рассматривать как С-строку и использовать в функциях, которые должны получать на вход С-строку.

7.6. Строки в стиле С

В языке С отсутствует строковый тип данных, поэтому строка представляется как **одномерный массив** типа char. Масив оканчивается нулевым байтом – «\0»

8. Объектно – ориентированное программирование

8.1. Суть ООП

Объект - часть памяти, которая используется для хранения значений.

Объекты имеют два основных компонента:

- **Свойства** (например: вес, цвет, размер, прочность, форма и т.д.).
- **Поведение**, которое они могут проявлять (например: открывать что-либо, делать что-то и т.д.).

ООП предоставляет возможность создавать объекты, которые объединяют свойства и поведение в самостоятельный союз, который затем можно многоразово использовать.

Это позволяет писать программы **модульным способом**, что упрощает не только написание и понимание кода, но и обеспечивает более высокую степень возможности повторного использования этого кода.

8.2. Понятия в ООП

ООП

Абстракция	Инкапсуляция	Модульность	Иерархия	Наследование	Полиморфизм

Абстракция – выделение общих характеристик объекта, исключая набор незначительных. С помощью принципа абстракции данных, данные преобразуются в объекты

Инкапсуляция – размещение в одном компоненте данных и методов, которые с ними работают. Также может означать скрытие внутренней реализации от других компонентов

Модульность – система состоит из независимых компонент – модулей, для которых определены программные интерфейсы их взаимодействие, а сами они построены по принципу «черного ящика» - их внутреннее содержание скрыто от внешнего пользователя

Иерархия – одни из компонент системы могут быть использованы как составные части для построения более сложных компонент. В иерархических системах используется также принцип рекурсии – в компоненту могут входить в качестве составных частей компоненты такого же типа (а в нее – аналогичные, и т.д. до бесконечности);

Наследование – позволяет описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом

Полиморфизм – возможность объектов с одинаковой спецификацией иметь различную реализацию. Базовый пример – перегрузка функций.

8.3. Классы

Класс - шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов. На практике ООП сводится к созданию некоторого количества классов, включая интерфейс и реализацию, и последующему их использованию.

class <имя класса> { <список членов класса> };

8.4. Методы

Метод - это функция или процедура, принадлежащая какому-то классу или объекту. Как и процедура в процедурном программировании, метод состоит из некоторого количества операторов для выполнения какого-то действия и имеет набор входных аргументов

8.5. Конструктор/деструктор

Конструктор класса — специальный блок инструкций, вызываемый при создании объекта.

Деструктор — это ещё один специальный тип метода класса, который выполняется при удалении объекта класса. В то время как конструкторы предназначены для инициализации класса, деструкторы предназначены для очистки памяти после него.

8.6. Доступ к членам класса

Член класса может быть:

- **приватным (private)** – это значит, что его имя может употребляться лишь внутри функций-членов класса и друзей класса, в котором этот член класса объявлен;
- **защищённым (protected)** – это значит, что его имя может употребляться лишь внутри функций-членов класса, друзей этого класса и производных от него классов;
- **публичным (public)** – это значит, что его имя может употребляться внутри любой функции (а также и вне функций в инициализаторах).

8.7. Шаблонные функции

Шаблоны функций — это функции, которые служат образцом для создания других подобных функций. Главная идея — создание функций без указания точного типа(ов) некоторых или всех переменных. Вместо этого мы определяем функцию, указывая тип параметра шаблона, который используется вместо любого типа данных. После того, как мы создали функцию с типом параметра шаблона, мы фактически создали «трафарет функции».

8.8. Шаблоны классов

Шаблоны классов работают точно так же, как и шаблоны функций: компилятор копирует шаблон класса, заменяя типы параметров шаблона класса на фактические (передаваемые) типы данных, а затем компилирует эту копию. Если у вас есть шаблон класса, но вы его не используете, то компилятор не будет его даже компилировать.

Шаблоны классов идеально подходят для реализации контейнерных классов, так как очень часто таким классам приходится работать с разными типами данных, а шаблоны позволяют это организовать в минимальном количестве кода. Хотя синтаксис несколько уродлив, и сообщения об ошибках иногда могут быть «объёмными», шаблоны классов действительно являются одним из лучших и наиболее полезных свойств языка C++.

9. Динамические структуры данных

9.1. Суть

Существует, однако, много задач, в которых требуются данные с более сложной (динамической) структурой. Для такой структуры характерно, что в процессе вычислений изменяются не только значения объектов, но и структура хранения информации. Поэтому такие объекты называются **динамическими информационными структурами**. Их компоненты, в свою очередь, на некотором уровне детализации представляют собой объекты со статической структурой, то есть они принадлежат к одному из основных типов данных.

Связный список является простейшим типом данных динамической структуры, состоящей из элементов (**узлов**). Каждый узел включает в себя в классическом варианте два поля:

- **данные** (в качестве данных может выступать переменная, объект класса или структуры и т. д.)
- **указатель** на следующий узел в списке.

9.2. Классификация

По количеству полей указателей:

- Связный список, содержащий только один указатель на следующий элемент, называется **односвязным.**
- Связный список, содержащий два поля указателя – на следующий элемент и на предыдущий, называется **двусвязным.**

По способу связи элементов

- Связный список, в котором, последний элемент указывает на NULL, называется **линейным.**
- Связный список, в котором последний элемент связан с первым, называется **циклическим.**

9.3. Виды

Односвязный линейный список (ОЛС).
Каждый узел ОЛС содержит 1 поле указателя на следующий узел. Поле указателя последнего узла содержит нулевое значение (указывает на NULL).

Односвязный циклический список (ОЦС).
Каждый узел ОЦС содержит 1 поле указателя на следующий узел. Поле указателя последнего узла содержит адрес первого узла (корня списка).

Двусвязный линейный список (ДЛС).
Каждый узел ДЛС содержит два поля указателей: на следующий и на предыдущий узел. Поле указателя на следующий узел последнего узла содержит нулевое значение (указывает на NULL). Поле указателя на предыдущий узел первого узла (корня списка) также содержит нулевое значение (указывает на NULL).

Двусвязный циклический список (ДЦС).
Каждый узел ДЦС содержит два поля указателей: на следующий и на предыдущий узел. Поле указателя на следующий узел последнего узла содержит адрес первого узла (корня списка). Поле указателя на предыдущий узел первого узла (корня списка) содержит адрес последнего узла.

Стеком называется упорядоченный набор элементов, в котором размещение новых и удаление существующих происходит с одного конца, называемого вершиной.

Очередью называется упорядоченный набор элементов, которые могут удаляться с её начала и помещаться в её конец.

Дерево – структура данных, представляющая собой древовидную структуру в виде набора связанных узлов.

Бинарное дерево — это конечное множество элементов, которое либо пусто, либо содержит элемент (корень), связанный с двумя различными бинарными деревьями, называемыми левым и правым поддеревьями. Каждый элемент бинарного дерева называется узлом. Связи между узлами дерева называются его ветвями.
Двоичная куча представляет собой полное бинарное дерево, для которого выполняется основное свойство кучи: приоритет каждой вершины больше приоритетов её потомков.

Граф – совокупность точек, соединенных линиями. Точки называются вершинами, или узлами, а линии – ребрами, или дугами.

9.4. Способы представления графов в программе

- матрица смежности;
- матрица инцидентности;
- список смежности (инцидентности);
- список ребер.

9.5. Обработка в программе

9.5.1. Алгоритмы обхода

- Поиск в ширину - подразумевает поуровневое исследование графа;
- Поиск в глубину - это алгоритм обхода вершин графа.

9.5.2. Поиск в ширину

I. Поиск кратчайшего пути в невзвешенном графе (ориентированном или неориентированном).
II. Поиск компонент связности.
III. Нахождения решения какой-либо задачи (игры) с наименьшим числом ходов.
IV. Найти все рёбра, лежащие на каком-либо кратчайшем пути между заданной парой вершин.
V. Найти все вершины, лежащие на каком-либо кратчайшем пути между заданной парой вершин.

9.5.3. Поиск в глубину

I. Поиск любого пути в графе.
II. Поиск лексикографически первого пути в графе.
III. Проверка, является ли одна вершина дерева предком другой.
IV. Поиск наименьшего общего предка.
V. Топологическая сортировка.
VI. Поиск компонент связности.