## Basics

### initialization/declaration

traditional

int x = 20;

int y;

uniform initialization

int x {20};

int y;

### OOPs

encapsulation

polymorphism

inheritance

abstraction & DRY

object

properties

behaviour



## class

by default attributes/functions are private

in a struct default is public

aggregate initialization

when all members of a class are public



## constructor

* special member function
* automatically called when object of class is created
* initialise data
* if any setups required for class/object to be used
* same name as of the class
* no return type

### default constructor

* no parameters
* or has parameters that have all default values
* no user provided initialization was provided

## functions

* reusable code
* return values
  + if no return value, then write void

ODR

### function overloading

based on num of parameters

or type of parameters

never based on return values

avoiding ambiguous

explicitly typecasting

mixing overloaded 7 default values

compile time

default values

to the right most

no named arguments

## polymorphism

operator overload

C++ python

function overload

C++ java

### virtual polymorphism

runtime

## compound types

composite

arrays

pointers

references

enum

class

struct

union

### references

* must be initialised (to a data)
* can not be bound to a const
* data type must be same
* no references to void
* can not be reseated (can not reference to another object)

std::reference\_wrapper

0000 0000 0000 0000 0000 0001 0100 1010

423577 423576 423575 423574

ptr = 423574 \*ptr = 330

qtr = 423574 \*qtr = ‘J’ 74

arr = 4000

ptr = arr = 4000

|  |  |  |
| --- | --- | --- |
| 4016 | 55 |  |
| 4012 | 44 |  |
| 4008 | 33 |  |
| 4004 | 22 |  |
| 4000 | 11 |  |
|  |  |  |

arr[i] = \*(arr + i\*4)

arr[3] = \*(arr + 3)

\*(4012)

44

3[arr] = \*(3 + arr)

\*(4012)

44

arr

brr

valgrind

## static

* created in static space
* retains value over function calls
* initialised to zero
* can not be initialised within class unless created as const
* will need a declaration/intialisation outside the class if not const
* belongs to class, rather than object
* can be accessed by class objects
* will not have a this pointer

### static member functions

* can only access static data
* can have any kind of data created locally
* can only call other static functions
* no static constructors
* not have any this pointer

## storage classes

static

extern

auto

register

mutable

## operator overloading

* operator

### can not overload

::

\*

.

?:

sizeof()

## inheritance

| Unix data members | public child | protected child | private child |
| --- | --- | --- | --- |
| public | public | protected | private |
| private | NI | NI | NI |
| protected | protected | protected | private |

overriding



diamond problem



Diamond problem

* resolved with virtual keyword

## Memory map of an application



mem classification

Data

Stack

Heap

Static

Text

Stack

Heap

Static

Text

## best practices

### declaration

uniform initialization

### functions

write return (even if it is void)

use const references wherever applicable

mention parameter names in function declarations

ODR

* a function/variable/type/template can have only one definition within a file

### class

class names with a capital letter

use struct for data-only structures

make member variables private, member functions public

(unless you have a good reason not to)

### constructors

if there are multiple constructors, use delegation

### strings

use std::string wherever possible

### references

place the ampersand next to the type (and not to the reference)

pass by const reference whenever possible

fundamentals types by value, pass class types by const reference

Herbert Schildt

The complete reference of C++

Scott Meyers

Effective Modern C++