TYPE-1

Cheatography

C++ Cheatsheet(inc. C++2011) Cheat Sheet

by Leupi via cheatography.com/5907/cs/1197/

Standard Sytemtypes and their range

int8_t	-128 to 127
uint8_t	0 to 255
int16_t	-32768 to 32767
uint16_t	0 to 65535
int32_t	-2147483648 to 2147483647
uint32_t	0 to 4294967295
int64_t	$-9.2 * 10^{18}$ to $9.2 * 10^{18}$
uint64_t	0 to $1.8 * 10^{19}$

Literals & Co.

255	Integer
0xaf	Hexadecimal Integer
1234.0	double
234.243f	float
true	bool
"Hello World"	string/c-string
'a'	char

Preprocessor

#include <LIB>	Includes standard header file LIB
#include "Header.h"	Includes Header.h file
#define PI 3.14159265359	Defines Easy Macro
#define f(a,b) a+b	More Complex Macro
#undef PI	Remove Definition
#ifdef PI	Checks for definition, if true, code will be compiled
#else	Else part of the above if-statement
#endif	Ends the if-statement

Nullpointer

```
int* pInt;
//old, deprecated:
if(pInt == NULL)
//new:
if(pInt == nullptr)
//still valid:
if(!pInt)
```

Bitwise Operators

&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Shift left
>>	Shift right

Boolean Logic

==	Test of equality
!=	Test of non-equality
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
!	NOT
&&	AND
||	OR

Boolean expressions in C++ are evaluated left-to-right!

Basic Operators

+	addition
-	subtraction
*	multiplication
/	division
%	modulo
++var	increase before evaluation
var++	increase after evaluation
condition ? result : alternative;	short form of if-like structure
::	Scope Operator
->	pointer-to-member
.	Access member
<< >>	Bitshift(with streams: input/output)

Pointers: Quick and dirty

&	Gets RAM adress of Variable(to save into pointer)
*	Dereferences pointer(returns it´s content) or defines a variable to be a pointer
->	Access pointer class member. same as (*pointer).member()
new	Create new object on heap, returns pointer to object
delete	Remove object at the pointer on heap

Pointers allocate space on heap, normal variables on stack!

using replaces typedef

```
//typedef is deprecated
typedef uint32_t uint32;
//now: using directive!
//using identifier = type_name;
using uint32 = uint32_t;
```

Auto Datatype

```
//auto is an automatic datatype:
int x = 4; //equals:
auto y = 4;
//works for most cases, esp. STL:
for(std::vector<int>::Iterator it = v.begin().....)
//with auto:
for(auto it = v.begin; it != v.end(); ++it)
```

Compile-time assertion check

```
static assert(bool_constexpr, string)
```

Multithreading

```
//Thread Creation & Management:
void thread_func(int a)
{
std::cout << "My Number is: " << a;
}
main()
{
std::thread t1(thread_func(1));
t1.join();
```


Sponsored by **Readability-Score.com**
Measure your website readability!
https://readability-score.com

Multithreading (cont)

```
}
//Synchronization via:
std::mutex.lock();
std::mutex.unlock();
```

Range based loops

```
//Easy range based loop:
std::vector<int> vec = .....;
for(int i : vec) //foreach i in vec
//Same with auto:
const auto vi = ....;
for(auto i : vi)
```

Works with all Containers with begin() and end()

Sponsored by **Readability-Score.com**
Measure your website readability!
https://readability-score.com

TYPE-2

Cheatography

CS 102 Exam 2 Cheat Sheet

by dtroutman63 via cheatography.com/44704/cs/13194/

Header

#include <iostream>

#include <iomanip>

#include <string>

#include <cmath>

#include <cstdlib>

#include <fstream>

#include <sstream>

#include <vector>

Function Prototypes

void functionName(const double[], int, int&);

int anotherName(int scores[][3], int size);

Function Calls

functionName(temperatures, size, high);

anotherName(scores, SIZE);

Problems

Subscript out of bounds

> Too many or few loops

No decimals

> Multiply by 1.0 or change double

error: expected unqualified-id before

> Use instance instead of type

String Manipulation

str.substr(pos, len)	returns string from pos
str.find(str1)	returns location index
str.find(str1, pos)	starts at pos then finds
str.insert(pos, str1)	inserts str1 at pos
str.erase(pos, length)	removes length at pos

Functions

Random #	(rand() % (max-min+1))+min
Power	pow(base, exp)
Abs Value Int	abs(num)
Abs Value Double	fabs(num)

OOP

vector	.push_back()
	.resize()
fstream	.open()
	.fail()
inserting filename	.c_srt()

Seeding the Function

```
cin>>seed;
srand(seed);
```

Switch

```
switch(trigger){
   case 0: break;
   case 1: break;
   default: break;
]
```

Bubble Sort for Array

```
int temp;
bool sorted = 0;
while(!sorted){
   sorted=1;
      for (i=1;i<num_elements;i++){
         if(arr[i-1]>arr[i]){
            temp=arr[i-1];
            arr[i-1]=arr[i];
            arr[i]=temp;
            sorted=0;
         }
      }
}
```

Infinite Loop

```
while (true){
   break;
}
```


Sponsored by **Readability-Score.com**
Measure your website readability!
https://readability-score.com

Insert and Erase / Fix

```
void fixSentence(string& sentence){
    int i = sentence.find(" ");
    string firstnoun;
    firstnoun = sentence.at(i+1);

    if (firstnoun == "a"&&"e"&&"i"&&"o"&&"u"){
        sentence.insert(1,"n");
    }

    else{
    int j = sentence.find(" ", i + 1);
    int k = sentence.find(" ", j + 1);
    int verblength = (k - (j+1));
    sentence.erase(j+1, verblength);
    sentence.insert(j+1, selectWord(VERBS, NUM));
    }
}
```

Extract from string stream

```
watch = "";
string tempstr;
sin>>day>>month>>year>>clouds>>temp>>wind>>watch;
while (sin>>tempstr){
   watch = watch + " " + tempstr;
}
```


Sponsored by **Readability-Score.com**
Measure your website readability!
https://readability-score.com

enum

```
enum Color {
    COLOR_RED, // assigned 0
    COLOR_BLUE // assigned 1
};
Color color=COLOR_RED;
// input/output
int input{};
std::cin >> input;
Color color=static_cast<Color>-
(input)
```

Prefer enum class if possible (C++11)

enum class (C++11)

```
enum class Color {
    RED,
    BLUE
};
Color color=Color::RED;
std::cout << static_cast<int>(c-
olor);
```

typedef

```
typedef double distance_t;
using distance_t = double; //
C++11
```

struct

```
struct Employee {
    short id;
    int age;
};
Employee joe = {1, 42}; //
initializer list
joe.age = 43; // member
selection operator
```

std::array - C++11

```
#include <array>
std::array<int, 3> tab {1, 2,
3};
tab[0] // or tab.at(0) with
bounds checking
tab.size() // length not
sizeof()
```

std::vector

```
#include <vector>
std::vector<int> tab {1, 2, 3,
4};
tab[0] // or tab.at(0) with
bounds checking
tab.size()
tab.resize(3); // expensive
// capacity & stack-like
behavior
std::vector<int> stack;
stack.reserve(5); // set
capacity
stack.push_back(3);
int x = stack.back(); // top of
stack
stack.pop_back();
```

capacity = allocated memory
length = active values
Setting capacity avoid resizing the array

std::tuple - C++11

```
#include <tuple>
// using namespace std;
tuple<int, double> t = make_t-
uple(3, 6.7);
int a = get<0>(t);
// std::tuple<int, double>
foo();
int a;
double b;
std::tie(a,b) = foo()
auto [a, b] = foo(); // C++17
```

Pointers

```
int value=5;
int *const p=&value; //const
pointer
const int value=5;
const int *p; // non-const
pointer to const int
// const pointer to const value
const int *const p=&value;
// int foo(int x);
int (*ptr)(int); // function
pointer
int (*const ptr)(int) = foo; //
const pointer
#include <functional>
std::function<int(int)> ptr =
foo; // C++11
ptr(5); // implicit dereference
or (*ptr)(5)
```

References

```
int value=5;
int &ref=value; // non-const ref
const int value=5;
const int &ref=value; // treats
value as const
```

References are similar to const pointers (cannot change address)

new & delete (dynamic allocation)

```
// scalar version
int *p=new int(3);
delete p;
// array version
int *p = new int[3] {1, 2, 3};
// C++11 syntax
delete[] p;
```


Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Classes

```
class Foo {
private:
    // non-static member initia-
lization
    int m_value = 10; // C++11
    // static member variable //
    static int m_id;
public:
    // constructor and
destructor
    Foo(int value):m_value(-
value);
    ~Foo();
   // copy constructor
    Foo(const Foo& foo)
                :m_value(foo.m_-
value);
    // static member function
    static int get_id(); // see
definition
    // const member function
    void print() const;
    // friend function
    friend void reset(Foo &foo);
};
Foo(6); // anonymous object
// static
int Foo::m_id=0;
int Foo::get_id() { return m_id;
} // no static
int id = Foo::get_id();
// const member function
void Foo::print() const {
whatever }
const Foo foo;
foo.print(); // no constness
violation
// friend function
```

Classes (cont)

```
friend void reset(Foo &foo) {
foo.m_value=0; }
```

1) no implicit default constructor if at least one constructor provided
2) implicit copy constructor and overloaded operator= use memberwise initialization (shallow copy)

std::initializer_list (C++11)

```
#include <initalizer_list>
int m_size;
int *m_array;
// prevent shallow copy (dynamic
mem alloc)
Foo(const Foo& foo) = delete;
Foo& operator=(const Foo& foo) =
delete;
Foo(const std::initializer_list-
<int>& list)
    :m_size(static_cast<int>(li-
st.size))
{
    // deep copy with for-each
loop
    int count=0;
    for (auto& value : list)
        m_array[count++] =
value;
}
Foo& operator=(const std::init-
ializer_list<int> &list)
{
    // same but delete[] m_array
    // if same size (no need for
new or delete)
    // deep copy
}
Foo foo { 1, 2, 3, 4 }; // call
constructor
Foo foo = { 1, 2, 3, 4 }; //
call operator=
```

For-each loops - C++11

```
for (int num : tab)
for (auto num : tab) // avoid
type conversion
for (auto &num : tab) // avoid
copy
for (const auto &num : tab) //
read-only
```

For-each loops don't work with array decayed to pointer (fixed or dynamic)

std::rand()

```
#include <cstdlib>
#include <ctime>
std::stand(static_cast<unsigned
int>(std::time(nullptr))); //
seed with time
std::rand(); // [0, RAND_MAX]
// [min, max]
return min + (std::rand()%(max--
min+1))
//better
static constexpr double fraction
{1.0/(RAND_MAX+1.0)};
return min + static_cast<int>((-
max-min+1.0)*(std::rand()*fract-
ion))
```

% is biased towards low numbers

Input validation with std::cin

```
// extraneous input
std::cin.ignore(32767, '\n'); //
clears buffer
// extraction failure
if (std::cin.fail()) { // type
or overflow
    std::cin.clear(); // back to
normal mode
    std::cin.ignore(32767,
'\n');
}
// switch-like validation
```


Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Input validation with std::cin (cont)

```
int x {};
do {
    std::cin >> x;
} while (x<0);
```

std::cerr and assert

```
#include <cstdlib> // for exit
std::cerr << "Error message";
exit(1);
#include <cassert>
assert(x==0); // combines cerr
and exit
assert(x==0 && "Error message");
// check at compile-time
static_assert(x==0, "Error
message"); // C++11
```

Operator overloading

```
int m_value;
int *m_array=nullptr;
// ret by value (ref to out of
scope var)
friend Foo operator+(const Foo
&foo, const Foo& bar);

// subscript []
// ret by ref to assign (l-
value)
int& operator[](const int idx)
{ return m_array[idx]; }

// overload int typecast
operator int() const { return
m_value; }
// std::cout
std::ostream& operator<<(s-
td::ostream& out, const Foo
&foo)
{
```

Operator overloading (cont)

```
    out << m_value;
    return out;
}
// std::cin
std::istream& operator>>(s-
td::istream& in, Foo &foo)
{
    in >> m_value;
    return in;
}
// increment operator as member
function
Foo& operator++(); // prefix,
++x
Foo operator++(int) // postfix,
x++
{
    Foo tmp(foo);
    // call implicit copy
constructor
    // or assignment with tmp =
foo
    // but beware of shallow
copy
    // increment this
    return tmp; // return by
value
}
```

1) unary op or if modif to left operand
-> member function
no modif, binary as friend or regular one
2) [], =, (), -> as member

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

TYPE-4

Cheatography

C++ references and pointers in function calls Cheat Sheet

by adamvm via cheatography.com/86080/cs/20075/

Regular variable

```
// variable definition


int count = 10;


// function definitions


void example( int i) {
    cout << i;
    i = 30;
}


int main() {
    example(count);
    cout << count;
    return 0;
}


// terminal output


10
10
```

Reference

```
// variable definition


int count = 10;


// functions definition


void example( int& i) {
    cout << i;
    i = 30;
}


int main() {
    example(count);
    cout << count;
    return 0;
}


// terminal output


10
30
```

Pointer

```
// variable definition


int count = 10;


// pointer definition


int* pCount = &count;


//functions definition


void example( int* i) {
    cout << *i;
    *i = 30;
}


int main() {
    example(pCount);
    cout << count;
    return 0;
}


// terminal output


10
30
```


Sponsored by **Readable.com**
Measure your website readability!
https://readable.com