C++ Programming I

Refresher

C++ Programming
February 22, 2018

Dr. P. Arnold
Bern University of Applied Sciences

Agenda

- Variables

Agenda

- Variables
- Data Types

Agenda

- Variables
- Data Types
- Keywords

Agenda

- ▶ Variables
- ▶ Data Types
- ▶ Keywords
- ▶ Compiling & Linking
  - ▶ PingPong
  - ▶ Inlcude Guards

Variables

Variables

Declaration, Initialisation and Definition

```
// Declaration
int x; // of variable int
int getValue(); // of function prototype

// Definition
int x; // same as declaration
int getValue(){ /* Definition */ } // without ';'

// Initialisation initialization is optional, but it's
// often a good programming practice
int x = 42; // refers to the "assignment" of a value

// initialization does not mean much for functions
```

- The variable type attribute tells the compiler the nature of data the variable can store, and the compiler reserves the necessary space for it
- The variable name is a friendly replacement for the address in the memory
- Use `camelCase` naming convention for variables
- Naming conventions differs for objects, functions etc.

Naming variables appropriately is important for writing good, understandable, and maintainable code!

Data Types

Fundamental C++ Variable Types

Data Type Ranges

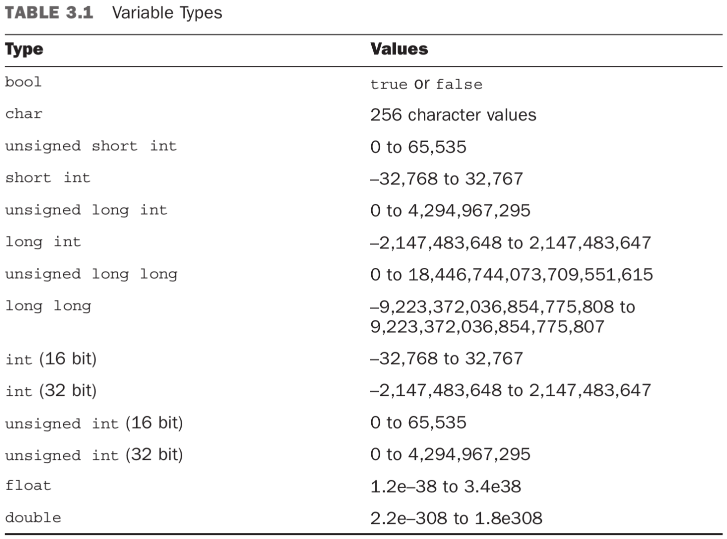


TABLE 3.1 Variable Types

Type	Values
`bool`	`true` or `false`
`char`	256 character values
`unsigned short int`	0 to 65,535
`short int`	–32,768 to 32,767
`unsigned long int`	0 to 4,294,967,295
`long int`	–2,147,483,648 to 2,147,483,647
`unsigned long long`	0 to 18,446,744,073,709,551,615
`long long`	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`int` (16 bit)	–32,768 to 32,767
`int` (32 bit)	–2,147,483,648 to 2,147,483,647
`unsigned int` (16 bit)	0 to 65,535
`unsigned int` (32 bit)	0 to 4,294,967,295
`float`	1.2e–38 to 3.4e38
`double`	2.2e–308 to 1.8e308

- Select correct data type according your needs!

Fundamental Types in C++

Determining the size of variables using `sizeof`

```cpp
#include <iostream>
using namespace std;

int main()
{
   cout << "Size of char : " << sizeof(char) << endl;
   cout << "Size of int : " << sizeof(int) << endl;
   cout << "Size of short int : " << sizeof(short int) << endl;
   cout << "Size of long int : " << sizeof(long int) << endl;
   cout << "Size of float : " << sizeof(float) << endl;
   cout << "Size of double : " << sizeof(double) << endl;
   cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;

   return 0;
}

// Output changes with compiler, hardware and OS
Size of char :       1
Size of int :        4
Size of short int :  2
Size of long int :   8
Size of float :      4
Size of double :     8
```

C++ 11 introduced fixed-width integer types! Include `<cstdint>` to use e.g. 8-bit signed and unsigned integers (`int8_t, uint8_t`)

Fundamental Types in C++
Limits

```
    std::cout << "char : "
              << int (std::numeric_limits<char>::min()) << ".."
              << int (std::numeric_limits<char>::max()) << "\n" ;

    std::cout << "int : "
              << std::numeric_limits<int>::min () << ".."
              << std::numeric_limits<int>::max() << "\n";

    std::cout << "short int : "
              << std::numeric_limits<short int>::min() << ".."
              << std::numeric_limits<short int>::max() << "\n";

    std::cout << "long int : "
              << std::numeric_limits<long int>::min () << ".."
              << std::numeric_limits<long int>::max () << "\n";

    std::cout << "float : "
              << std::numeric_limits<float>::min () << ".."
              << std::numeric_limits<float>::max () << "\n";

    std::cout << "double : "
              << std::numeric_limits<double>::min () << ".."
              << std::numeric_limits<double>::max () << "\n";

    std::cout << "wchar_t : "
              << std::numeric_limits<wchar_t>::min () << ".."
              << std::numeric_limits<wchar_t>::max ();
```

Fundamental Types in C++

Limits

```
char :       -128..127
int :        -2147483648..2147483647
short int : -32768..32767
long int :  -9223372036854775808..9223372036854775807
float :      1.17549e-38..3.40282e+38
double :     2.22507e-308..1.79769e+308
wchar_t :   -2147483648..2147483647
```

- Size of type in bytes.

Tip:

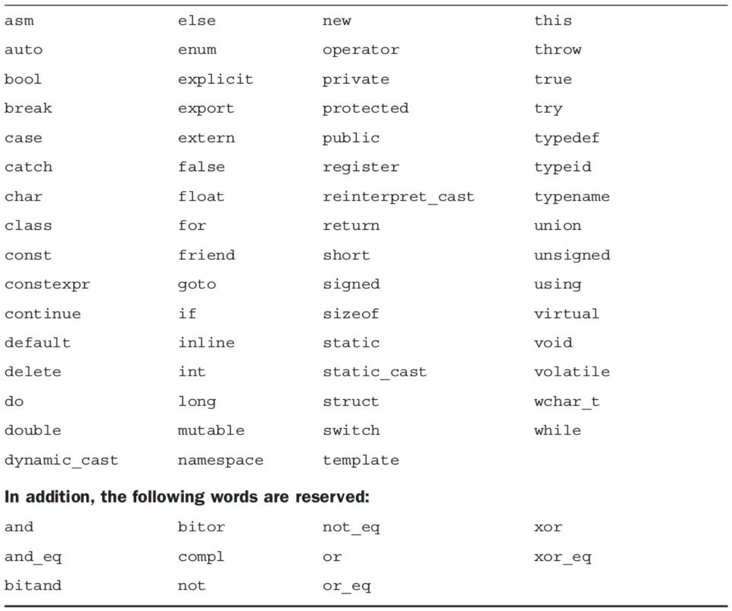
Include <limits> from the standard library

Keywords

Bern University
of Applied Sciences

Major Keywords

reserved by C++



asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
constexpr	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

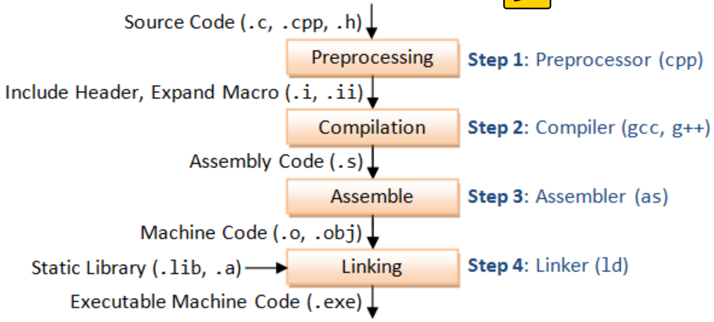
In addition, the following words are reserved:

and	bitor	not_eq	xor
and_eq	compl	or	xor_eq
bitand	not	or_eq	

Compiling & Linking

Compiling an Linking

GCC Compilation Process



Source Code (.c, .cpp, .h)
Preprocessing
Step 1: Preprocessor (cpp)
Include Header, Expand Macro (.i, .ii)
Compilation
Step 2: Compiler (gcc, g++)
Assembly Code (.s)
Assemble
Step 3: Assembler (as)
Machine Code (.o, .obj)
Static Library (.lib, .a)
Linking
Step 4: Linker (ld)
Executable Machine Code (.exe)

- Compile time
- Link time
- Run time

Compiling an Linking

Order of Compilation

```
void ping(int n_times)
{
    std::cout << "ping: " << n_times << std::endl;
    if(n_times > 0)
    {
        pong(--n_times);
    }
}

void pong(int n_times)
{
    std::cout << "pong: " << n_times << std::endl;
    if(n_times > 0)
    {
        ping(--n_times);
    }
}
```

- Do you see a problem?

Compiling an Linking

Order of Compilation

```
void ping(int n_times)
{
    std::cout << "ping: " << n_times << std::endl;
    if(n_times > 0)
    {
        pong(--n_times);
    }
}

void pong(int n_times)
{
    std::cout << "pong: " << n_times << std::endl;
    if(n_times > 0)
    {
        ping(--n_times);
    }
}
```

- Do you see a problem?
- This code wont compile! Why?

Compiling an Linking
Order of Compilation

```cpp
void ping(int n_times)
{
    std::cout << "ping: " << n_times << std::endl;
    if(n_times > 0)
    {
        pong(--n_times);
    }
}

void pong(int n_times)
{
    std::cout << "pong: " << n_times << std::endl;
    if(n_times > 0)
    {
        ping(--n_times);
    }
}
```

- Do you see a problem?
- This code wont compile! Why?
- `pong` not declared when compiling `ping`

Compiling an Linking

Forward Declaration

```
// forward declaration
void ping(int n_times);
void pong(int n_times);

void ping(int n_times)
{
    std::cout << "ping: " << n_times << std::endl;
    if(n_times > 0)
    {
        pong(--n_times);
    }
}

void pong(int n_times)
{
    std::cout << "pong: " << n_times << std::endl;
    if (n_times > 0)
    {
        ping(--n_times);
    }
}
```

- Use forward declaration!

Compiling an Linking

Seperate Implementation - Header File

```
#ifndef PINGPONG_H
#define PINGPONG_H

void ping(int n_times);
void pong(int n_times);

#endif // PINGPONG_H
```

- Declaration of function `ping` and `pong`
- Visible to the user
- Note the include guards

Compiling an Linking

Seperate Implementation - Source

```
#include "iostream"
#include "pingpong.h"

void ping(int n_times)
{
    std::cout << "ping: " << n_times << std::endl;
    if(n_times > 0)
    {
        pong(--n_times);
    }
}

void pong(int n_times)
{
    std::cout << "pong: " << n_times << std::endl;
    if(n_times > 0)
    {
        ping(--n_times);
    }
}
```

- Definition of function `ping` and `pong`
- Can be hidden from the user!
- Can be a binary file

Compiling an Linking

Seperate Implementation

```
#include <iostream>
#include "pingpong.h"

int main ( )
{
    std::cout << "Lets play!" << std::endl;
    ping(10);

    std::cout << "Next round..." << std::endl;
    pong(5);

    return 0;
}
```

- Functions are included: `#include pingpong.h`
- What is the output? (Demo)

Note

Include system libraries with <SysLib.h> and user libraries with “UserLib.h”

Include Guard

Macro

```cpp
#ifndef INCLUDEHEADER1_H
#define INCLUDEHEADER1_H

#include "includeHeader2.h"


#endif // INCLUDEHEADER1_H
```

```cpp
#ifndef INCLUDEHEADER2_H
#define INCLUDEHEADER2_H

#include "includeHeader1.h"

#endif // INCLUDEHEADER2_H
```

- Multiple is a problem of recursive nature for the preprocessor
- One of the most frequently used macro-based functionality in C++
- PingPongGame **Demo**

Thank You

Questions

???