The C++ Environment

Overview

- Short History of C and C++
- Steps for a C++ Program
- Structure of a C++ Program
- Conventions and Standards
- Placing Comments
- Compiling and Linking

2

Objectives

This unit is meant as an introduction to the C language. Our aim is to give a reasonably but thorough introduction to the “procedural” elements of the language.

The topics discussed in this Unit
• Introduction
• Basic concepts
• The lay-out of your C program.
• Comments in a C program

Introduction

C is the most common used program language for system developers, but can be used for a wide range of applications. More and more people turn to the C programming language for this reason. The C programming language is often called a 'middle level language' because in C is possible to work on a lower level than in higher level languages as Pascal, Modula 2 and Fortran.

C is however a language with certain pitfalls that is the programmer can easily create a bug if he (or she) is not very accurate with the layout of the program. It is rather easy to make a mistake that can cause the computer to lock-up or cause memory problems.

This chapter will handle the basic concepts you need to know before creating your own C program.

.

History of C

- Developed by Kernighan and Ritchie (K&R)
- Developed for compatibility reasons
- Developed at the same time as UNIX (1969-1973)
- ANSI developed an international standard (1989), adopted by ISO (1990)
- Standards: C89/C90 and C99

3

Short History

C is developed because there was a need for a powerful language that was transferable to different systems. This means that a program can be converted to work on different machines with as little changes to the code as possible. This is a contrast with programs written in assembly code. Assembly code is very machine dependent these programs need to be completely rewritten to work on a different type of system.
C also allows us to create more complex and efficient programs.
C was first used for a transferable operating system UNIX. Most today used programs are written in C although C++ is marching up.
Dennis Ritchie is the brain behind the language C. The definition of the C-language was first given by Brian Kernighan and Dennis Ritchie in the book The C Programming Language.
Later the ANSI committee created a universal standard for this language. This ANSI standard was finished in 1989.
When creating programs it is important to make the code compliant with this standard so that it is compatible with all compilers that support the ANSI standard.

History of C++

- Developed by Bjarne Stroustrup
- Adds object-oriented and generic programming constructs to C
- Started in 1979, known as C++ since 1983, since 1998 an ISO standard
- Standards C++98, C++03, C++0x (future, end 2011)

4

Short History

Development of C++ was started in 1979 by Bjarne Stroustrup as an enhancement to the C language. Originally it was called “C with Classes” and in 1983 it was renamed to C++.

Enhancements to the C language are object oriented programming, operator overloading and later exception handling and templates (generic programming). The first ISO standard was ratified in 1998 and updated in 2003. The upcoming 2011 standard (formerly known as C++0x) will add many new features like lambda functions and move semantics.

.

What Makes the C/C++ Program

- C/C++ Program is the executable Program
- C/C++ Code divided into
  - Source files
  - Header files
- Source files contain C/C++ Code
- Header files contain common information
  - Declarations
  - Included in source file(s)

5

Basic Concepts

The objective of this section is to write a very simple C program. When we refer to a program we refer to the executable. When creating a program first you type in the code (source) which you store as a text file that usually has the extension .c. This source file is then compiled and linked to create the executable to which we will refer as the program. Whenever we want to change a part of the program the source file needs to be modified and compiled to create a new executable program.

Steps for a C/C++ Program

- Create header and source files
- Use compiler to translate the code
- Link the compiled code into an executable file
- Run the program

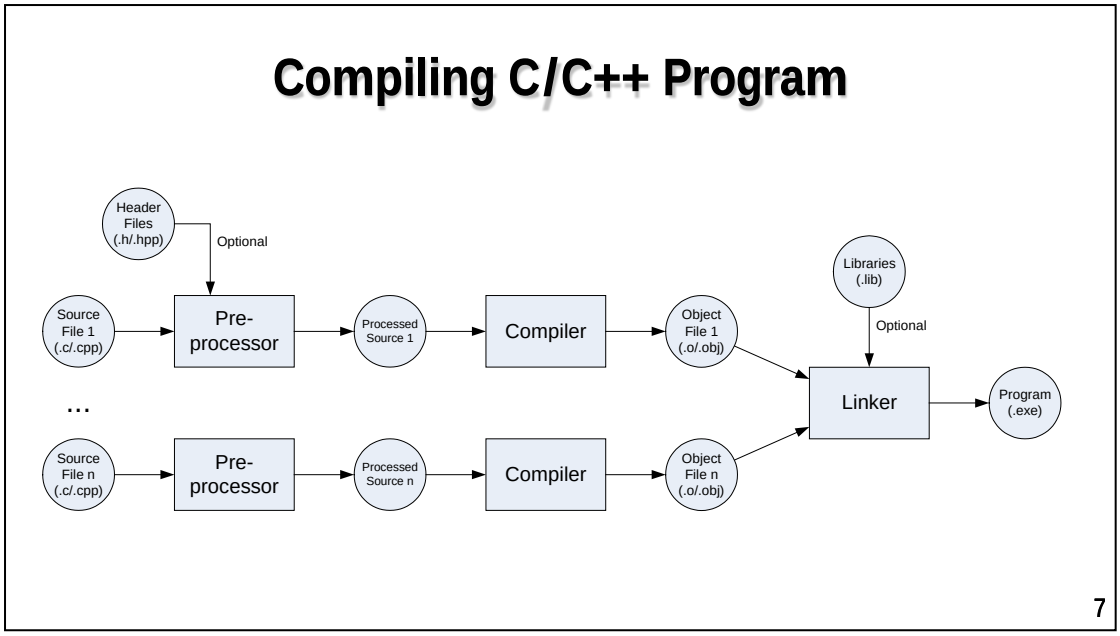
6

Steps

When a source file is compiled the compiler checks the code against the syntax of C and if no errors occur it creates an intermediate file, an object file (.obj). This object file is fed to the linker which adds some basic functionality to the file and creates the executable file (.exe).

It is possible to divide your source file in several files which need to be compiled separately. All the compiled files (.obj) are all fed to the linker which resolves references between the files and creates the final executable.

.



Source Files

Are plain text files containing C/C++ Code. A C program can be created from one or more source files.

Header Files

Are plain text files usually containing declarations (e.g. from a library) that a source file uses.

Pre-processor

The job of the pre-processor is to parse the source file for pre-processor instructions. The pre-processor changes the source file by for example including header files, expanding macros and removing code due to conditional compilation constructs.

Compiler

The first job of the compiler is to check the C code against the C syntax. If the compiler detects an error (compiler error) the compiler terminates with an error message. If the code is correct the compiler translates the code to an object file.

Linker

It is possible to have a reference in one source file to some function or variable in another source file or library. The linker solves these references and issues a warning if the reference cannot be solved.

Project

The project is compiler dependent. Most compilers use a so called project. This project has all settings for the compiler and linker. Projects typically include information about:

- Compiler flags
- Linker flags
- Source files to be compiler
- Environment

Some compilers do not use a project but a makefile. First consult your compiler manual to see what your compiler uses.

Structure of a C/C++ Source

- Consists of
  - Functions
  - Variables
- Functions contain statements that describe actions to take
- Variables contain values used in computations

8

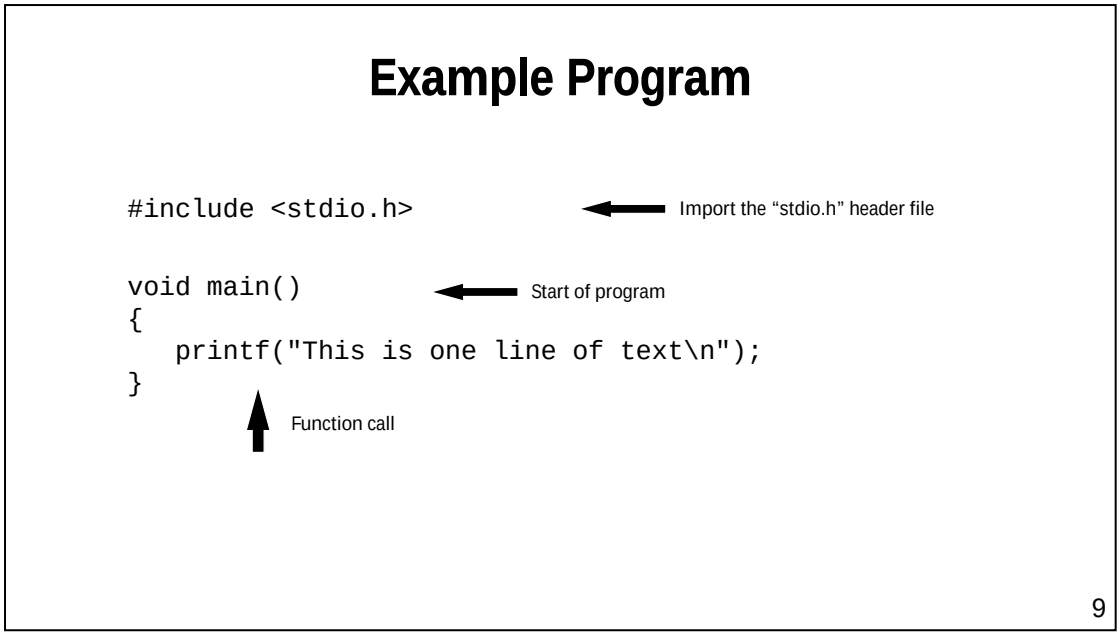
Structure of a C Program

The C source files consist of a number of functions and variables. All programs must contain the function *main()*. This function designates the start of a program. After the program starts the rest of the program will be a chain reaction of function calls (execution of discrete blocks of code).

The best way to set up a source file is to divide the entire source in to small discrete blocks of code called functions and execute these when needed. These discrete blocks of code can use and create variables for use in calculations or operations.

In order to have a clear source file it is we need to split each discrete block and even the whole source file into two discrete blocks of code. These two blocks are one information block and one actual code block. In C we always have to say what we are going to use (the information block) and then use it.

.



Header File

The statement `#include <stdio.h>` is a command that tells the compiler (actually pre-processor) to get the file *stdio.h* and include it when compiling this source file. This file is called the header file and mostly contains common information about constants and functions.

main()

main() is a function. This function is the start of your program. Every program must have one and only one *main()* function. If your program is created by using several source files you must make sure that only one source file has a main function. Later more on functions.

Placing Comments

- Comment starts with: /*
- Ends with: */
- No nested comments
- Can be placed where whitespace is allowed
- Place them to increase the readability of the program
- In C++ single line comment with: //

10

Comments

Most C programmers tend to create very cryptic and unreadable source files. A lot of programmers do not know what they have created a day earlier if they look at their code. This is why it is so important to always comment what you are doing. In C comments are placed between /* and */. All text between these symbols will be discarded by the compiler. Comments can be placed anywhere in the source file where whitespace is allowed. Whitespace are the spaces newlines and tabs.

```
#include <stdio.h>

/* start main */
void main()
{
	int counter; /* counter variable */

	printf("Some text to print\n");/* print the text */
}
```

Example of Illegal comments

Placing comment is allowed anywhere in the source file it is however not allowed to have nested comments.

```
#include <stdio.h>

void main()
{/* start main */
	int counter; /* counter variable */

/*
	printf("Some text to print\n");/* print the text */
*/
}
```

.

Lay-Out of Source File

- Create a readable source file
- Use comments to clarify the code
- Use newlines and spaces(whitespace) to create a standard look and feel
- Indent each new block

11

Comment Your Program

As mentioned earlier a source file can become very unreadable. In order to create source files in a consistent way most companies determine a so called style-guide. This guide has a few rules for the standard look of a source file.

Besides these style guides it is always good to comment blocks of code in the source.

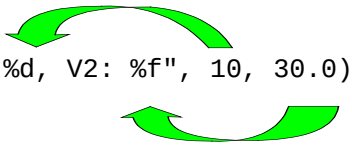
```
/*
       Explanation of code to come, or a general description of
       the functions in this source file.
*/

#include <stdio.h>

void main()
{
       /* Indent each new block */
       printf("The C Language");/* Indent inside functions */
       printf("This source file looks nice\n");
}
```

Simple *printf()* Statements

- Use *printf()* to send information to the screen
- Uses format string
- Printing text
  - `printf("String")`
- Printing values
  - `printf("V1: %d, V2: %f", 10, 30.0);`



12

Printing Values

Printing in C is done by using the *printf()* function. This function uses a format string to specify what to print. Normal characters in the string are translated to normal characters and will be printed. The following example will print a simple string on the screen.

```
printf("This is a C course");
```

Everything between the "" will be printed on the screen. This format string can also contain special specifiers to print values on the screen. The '%' followed by a type tells the *printf()* function that following the format string is a value that needs to be printed on the place of the '%' in the string. The character after the '%' specifies what kind of value to print.

```
printf("An integer value : %d", 10);
```

It is possible to print more than one value by using the '%' sign every time and passing enough values to the *printf()* function using a ','.

```
printf("Value 1 : %d, Value 2 : %f", 10, 30.0);
```

The specifier after the '%' must be the same type as the value we want to print. If these two types do not match the output will be not what we would expect.

Using character constants it is able to add new line characters to the format string.

```
print("The first line.\nThe second line");
```

output:

```
The first line.
The second line.
```

The next page has a table of the different types of values and a table of the escape characters we can use in the format string.

Table of *printf* codes

Sign	Type	Output
%c	int or char	Single character
%d or %i	int	Signed decimal integer
%o	int	Unsigned octal integer
%u	int	Unsigned decimal integer
%x	int	Unsigned hexadecimal integer using "abcdef"
%X	int	Unsigned hexadecimal integer using "ABCDEF"
%e or %E	double	Signed value having the form [–]d.dddd **e** [sign]dd[d] where d is a single decimal digit, dddd is one or more decimal digits, dd[d] is two or three decimal digits depending on the output format and size of the exponent, and sign is + or –. %e uses lower case **e** in the output while %E uses upper case **E** in the output.
%f	double	Signed value having the form [–]dddd.dddd, where dddd is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
%g or %G	double	Signed value printed in **f** or **e** format, whichever is more compact for the given value and precision. The **e** format is used only when the exponent of the value is less than –4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it. %G uses upper case **E** instead of lower case **e**.
%a or %A	double	Signed hexadecimal double precision floating point value having the form [−]0xh.hhhh **p**±dd, where h.hhhh are the hex digits (using lower case letters) of the mantissa, and dd are one or more digits for the exponent. The precision specifies the number of digits after the point. %a uses "abcdef". %A uses "ABCDEF".
n	int*	Number of characters written so far. This value is stored in the integer whose address is given as the argument.
p	void*	Prints the argument as an address in hexadecimal digits.
s	char*	Print the characters up to the first \o character.

Table of escape sequences

Escape sequence	Represents
\a	Bell (alert)
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\"	Double quotation mark
\\	Backslash
\?	Question mark
\ooo	ASCII character in octal notation
\xhh	ASCII character in hexadecimal notation
\xhhhh	Unicode character in hexadecimal notation