

Programming in C++ without the Main () Function

Gundeep Singh Bindra

Abstract—Research into the cognitive aspects of programming originated in the study of professional programmers (either experts or those learning to program). As personal computers become widespread, and most new domestic appliances incorporate microprocessors, many more people are engaging in programming-like activities. Some of these are studied as “end-user” programmers, by analogy to professional programming, but many encounter tasks and contexts completely unlike conventional programming but developing and analyzing atypical program source codes. The nature and extent of programming activity is changing rapidly. In the domain of professional programming, new tools continue to change the day-to-day nature of design and coding. This paper puts forth the source code of a program which defines an out-of-the-box feature to reach a defined objective.

Keywords—programming; atypical; no main; #define; review; #decode; trick.

I. INTRODUCTION

C++ (pronounced "see plus plus") [2] is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as an intermediate-level language, as it comprises a combination of both high-level and low-level language features [3]. It was developed by Bjarne Stroustrup starting in 1979 at Bell Labs as an enhancement to the C language and originally named C with Classes. It was renamed C++ in 1983 [4]. C++ is one of the most popular programming languages [5][6] and its application domains include systems software (such as Microsoft Windows), application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games [7]. Several groups provide both free and proprietary C++ compiler software, including the GNU Project, Microsoft, Intel and Embarcadero Technologies. C++ has greatly influenced many other popular programming languages, most notably C# and Java. C++ is also used for hardware design, where the design is initially described in C++, then analyzed, architecturally constrained, and scheduled to create a register-transfer level hardware description language via high-level synthesis [8].

The language began as enhancements to C, first adding classes, then virtual functions, operator overloading, multiple inheritance, templates, and exception handling among other features. As a programming language, C is rather like Pascal or Fortran. Values are stored in variables. Programs are structured by defining and calling functions. Program flow is

controlled using loops, if statements and function calls. Input and output can be directed to the terminal or to files. Related data can be stored together in arrays or structures [9]. Of the three languages, C allows the most precise control of input and output. C is also rather more terse than Fortran or Pascal. This can result in short efficient programs, where the programmer has made wise use of C's range of powerful operators. It also allows the programmer to produce programs which are impossible to understand. Programmers who are familiar with the use of pointers (or indirect addressing, to use the correct term) will welcome the ease of use compared with some other languages. Undisciplined use of pointers can lead to errors which are very hard to trace. This course only deals with the simplest applications of pointers. It is hoped that newcomers will find C a useful and friendly language. Care must be taken in using C. Many of the extra facilities which it offers can lead to extra types of programming error. You will have to learn to deal with these to successfully make the transition to being a C programmer.

II. DEFINING THE TERM “PROGRAMMING”

A. Historic definitions

Early publications in computing were written by practitioners, building and applying computers to practical tasks (though they often worked in a university context). These practitioners offered clear definitions of programming, without necessarily intending them as deep analyses of human activity. Nevertheless, these early publications are now seldom cited, and it is instructive to quote from them.

Hartree (1950) explained “The process of preparing a calculation for a machine can be broken down into two parts, ‘programming’ and ‘coding’. Programming is the process of drawing up the schedule of the sequence of individual operations required to carry out the calculation” (Hartree 1950, p. 111). Coding was of course extremely time-consuming before the development of assembler languages, but programming soon became the predominant activity. Wilkes (1956) made the distinction between the programme and other forms of calculation as follows: “The sequence of orders is known as the programme, and the machine performs it automatically without intervention from the user” (Wilkes 1956, p. 2).

“Programming is basically a process of translating from the language convenient to human beings to the language convenient to the computer” (McCracken 1957). Programming is the “spadework” of finding a precise mathematical formulation and method of solution, possibly notated in a “convenient problem-oriented language” whose symbols are “more closely related to the mathematical problem to be solved” [9.].

G. S. Bindra is with the Department of Computer Science, SRM University, NCR Campus, New Delhi 201204 India (phone: +91-9711944200 +91-9997777057 ; fax: +91-135-2658605; website: <http://gundeepbindra.com> e-mail: mailbox@gundeepbindra.com ;).

B. Research definitions

An introductory chapter to the book “Psychology of Programming” notes that the programming has changed from “describing calculations” to “defining functions”, and then “defining and treating objects”. These terms reflect the changes in programming from mathematical to more general data processing issues. Within mainstream computer science (i.e. outside the psychology of programming field), these advances may be considered more or less elegant or intuitive by computer scientists (Blackwell & Hague 2001b), but the criteria for defining these terms are often (as in mathematics), a matter for polite agreement rather than empirical investigation. The agreed basis of the human activity of programming is still, as defined in Collins dictionary, “to write a sequence of coded instructions fed into a computer ... to arrange data in a suitable form so that it can be processed by a computer ... to feed a program into a computer”[9].

III. OBJECTIVES

1. To explore the possibilities of having a program which runs without a main() function that is recognized by the compiler.
2. The supplementary objective of this paper is to even build a flair for developing atypical programs and stimulate out of the box thinking.

IV. NEED OF THE STUDY

One day on my usual Data Structures practical class , I was trying to compile and run my program on my system , surprisingly after checking the program I realized that I committed a mistake that was a crime in computer programming, I forgot to include the main() function from then I started working on the possibility of having a program without a main() function. Yes, there can be a C program without a main function and started my findings, I came through very interesting functions which besides developing and increasing my keen interest in the subject, also added to my knowledge.

Besides all what I said on how it started, I would like to emphasize the fact that programming essentially includes out of the box thinking and gives a lot of room for individual exploring many less known features.

This paper aims at increasing the horizons of many programmers and help them realize that a lot of out of the programs can be developed and build and there are many less prominent but interesting features.

V. MAIN() FUNCTION

In many programming languages, the main function is where a program starts execution. It is responsible for the high-level organization of the program's functionality, and typically has access to the command arguments given to the program when it was executed[2].

The main function is generally the first programmer-written function run when a program starts, and is invoked directly

from the system-specific initialization contained in crt0 or equivalent. However, some languages can execute user-written functions before main runs, such as the constructors of C++ global objects.

VI. SOURCE CODE

A. Prototype I

1) Version 1.0

```
#define prototypeone main
#include <stdio.h>
int prototypeone()
{
    printf("Prototype 1: Version 1.0 !\n");
    return 0;
}
```

The above code works perfectly on Visual C++ but I am not very sure that it will be supported by all the compilers.

2) Version 1.1

```
#define prototypeone(a,b,c,d) d##c##b##a
int prototypeone(n,i,a,m) (void)
{
    printf("Prototype 1: Version 1.1 !\n");
}
```

3) Findings

You can use the #define directive to give a meaningful name to a constant in your program.

The #define[10.] directive substitutes token-string for all subsequent occurrences of an identifier in the source file. The identifier is replaced only when it forms a token. For instance, identifier is not replaced if it appears in a comment, within a string, or as part of a longer identifier.

A #define without a token-string removes occurrences of identifier from the source file. The identifier remains defined and can be tested using the #if defined and #ifdef directives.

The two forms of the syntax are:

1. #define identifier token-string_{opt}
2. #define identifier[(identifier_{opt}, ... , identifier_{opt})] token-string_{opt}

In our case I would simply define begin as an appropriate replacement of main by begin. Following is the code:

```
#define main begin
```

The #undef directive causes an identifier's preprocessor definition to be forgotten.

Hence now we understand that the program still has a main function. If you trace through what the #define's are doing it's just changing "begin()" to "main()". The compiler still sees and compiles "main()".

B. Prototype II

1) Version 1.0

```

_start()
{
    printf(" Prototype 2: Version 2.0 !");
}

```

Compilation: ~ Gundeep\$ gcc -nomain <filename.c>

2) Version 2.1

```

_start()
{
    prototypetwo();
}

prototypetwo()
{
    printf("Prototype 2: Version 2.1 !\n");
    exit(0);
}

```

Compilation: ~ Gundeep\$ gcc est.c -nostdlib -lc

3) Version 2.2

```

#include<stdio.h>
#include <unistd.h>

_start()
{
    _exit(prototypetwo());
}

int prototypetwo(void)
{
    printf("Prototype 2: Version 2.2 !\n");
    return 42;
}

```

Compilation:

```

~ Gundeep$ gcc -O3 -nostartfiles <filename.c>
~ Gundeep$ ./a.out
Hello
~ Gundeep$ echo $?
42

```

C. Prototype III

1) Version 1.0

We use a little C++ trick to rename our main() function to

prototypethree().

Here's the code of the prototype III without a main function.

```

#include<stdio.h>
#define decipher(g,u,n,d,e,e,p) g##p###n###d
#define prototypethree decipher(m,b,i,n,d,r,a)
char *program=" #include<stdio.h>%cchar*
program=%c%s%c;%cint begin()%c{%cprintf
program,10,34,program,34,10, 10,10,10);%c}";

int prototypethree()
{
    printf(" Hi , This logic worked for me !!");
    printf(program,10,34,program,34,10,10,10,10);
}

```

2) Findings

Here we are using preprocessor directive #define with arguments. A Preprocessor is a program which processes the source code before compilation. The '##' operator is called the token pasting or token merging operator. That is we can merge two or more characters with it.

The second line of the program.

```
#define decipher(g,u,n,d,e,e,p) g##p###n###d
```

The macro decipher(g,u,n,d,e,e,p) is being expanded as "gpnd" (The ## operator merges g,p,n & d into gpnd). The logic is when you pass (g,u,n,d,e,e,p) as argument it merges the 4th,1st,3rd & the 2nd characters(tokens).

Now, the third line of the program.

```
#define prototypethree decipher(m,b,i,n,d,r,a)
```

Here the preprocessor replaces the macro "prototypethree" with the expansion decipher(m,b,i,n,d,r,a). According to the macro definition in the previous line the argument must be expanded so that the 4th,1st,3rd & the 2nd characters must be merged. In the argument (m,b,i,n,d,r,a) 4th,1st,3rd & the 2nd characters are 'm','a','i' & 'n'.

So the third line "int prototypethree()" is replaced by "int main()" by the preprocessor before the program is passed on for the compiler. That's it.

VII. CONCLUSION

The bottom line is there can never exist a C program without a main() function. A program must have a main function (compulsorily). Without a main() function the compilation of the source code is impossible. Here, in all prototype and various versions, we are just playing a gimmick that makes us believe that the program runs without the main function. But here we are using the preprocessor directive to intelligently replace the words prototypeone, prototypetwo and prototypethree by "main" in different illustrations.

In simple words:

```

int prototypone()=int prototypetwo()
=int prototypethree()=int main().

```

ACKNOWLEDGMENT

At the outset, in all humility, I thank the Almighty for the plentiful blessings He has showered on me to undertake and

see through to completion this paper. Words are inadequate when I wish to thank my parents and my sister Harshana , who have always taken pride in all my academic and co- curricular pursuits. I wish to gratefully acknowledge the support of my teachers who have been the pillar of my endeavors.

It would not be out of place to thank all the authors and researchers whose work I have consulted.. My thanks are due to the founders of the various softwares that I have used in writing this paper.

REFERENCES

- [1] Wikipedia "The Free Encyclopedia". Accessed: March 9, 2011
- [2] Herbert Schildt (1998-08-01). C++ The Complete Reference Third Edition. Osborne McGraw-Hill. ISBN 978-0078824760. Accessed: March 19, 2011
- [3] Bjarne Stroustrup (2010-03-07). "C++ Faq: When was C++ Invented". ATT.com. http://www2.research.att.com/~bs/bs_faq.html#invention. Retrieved 2010-09-16. Accessed: April 15, 2011
- [4] "Programming Language Popularity". 2009. <http://www.langpop.com>. Retrieved 2009-01-16. Accessed: June 2, 2011
- [5] "TIOBE Programming Community Index". 2009. <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. Retrieved 2009-05-06. Accessed: April 15, 2011
- [6] C++ Applications. <http://www2.research.att.com/~bs/applications.html>. Modified June 18, 2011. Accessed: April 1, 2011
- [7] "What's CvSDL?". <http://www.cvsdl.com/>. Retrieved 2010-03-08.
- [8] About C, "ASIC World". http://www.asic-world.com/scripting/about_c.html. Published: May-28-2011. Accessed: April 12, 2010.
- [9] Alan F. Blackwell, University of Cambridge Computer Laboratory "What is Programming?" Published: 14th Workshop of the Psychology of Programming Interest Group, Brunel University, June 2002 www.ppig.org. In J. Kuljis, L. Baldwin & R. Scoble (Eds). Proc. PPIG 14 Pages 204-218. Accessed: April 12, 2010.
- [10] MSDN Library, "The #define Directive - Visual Studio 2005" [http://msdn.microsoft.com/en-us/library/teas0593\(v=VS.80\).aspx](http://msdn.microsoft.com/en-us/library/teas0593(v=VS.80).aspx). Accessed: April 12, 2010.
- [11] Paul Deitel, Harvey M. Deitel, "C++: How to program". (7th Edition) Book. Publisher: Prentice Hall, 2009. ISBN: 0136117260. 1068 pages.
- [12] Nicolai M. Josuttis, "The C++ Standard Library: A Tutorial and Reference". Book. Publisher: Addison-Wesley Professional. Published: August 12, 1999. ISBN-10: 0-201-37926-0. 832 Pages.
- [13] Bjarne Stroustrup, "The C++ Programming Language: Special Edition". Book. Publisher: Addison-Wesley. Published: 11 February 2000. ISBN 0201700735. 911 pages.
- [14] Margaret A. Ellis,"The Annotated C++ Reference Manual". Book. Publisher: Addison-Wesley Professional. Published: January 11th 1990. 446 pages.
- [15] Allen Downey, "How to Think Like a Computer Scientist: C++ Version". Publisher: Green Tea Press. Published: 1999 189 pages.