The history of C

Andy Verkeyn
30 september 1999
2nd version: 16 october 2002

History overview

early 1960 (+/- 1963)	CPL (Combined Programming Language) This language was designed by Cambridge and London University and therefore became known under the nickname Cambridge Plus London. It was developed to be able to cope with all possible kind of problems (like Ada) but was actually never fully implemented.
mid 1960	The Multics Project MIT, General Electronic and Bell Labs jointly started a new project. Their goal was to write the first experimental time-sharing operating system. As an implementation language they used PL/I.
1965	TMG (Trans MoGrifiers) This language, created by McClure, was specially developed to ease the writing of compilers. It was used by Bob Morris and Doug McIlroy to create a PL/I compiler for Multics.
late 1960 (+/- 1967)	BCPL (Basic CPL) Martin Richards (Cambridge University) created this simplified version of CPL while visiting MIT (Massachussets Institute of Technology). This language was used in Cambridge for many years, including for some very interesting projects (also in Oxford and PARC). It was only in the late 1980's that BCPL was replaced by C in its own home labs.
1968-1969	Because Bell Labs started to realize that the promise of the Multics project could only be fulfilled too late and too expensively, they first started an informal group, led by Ken Thompson, to investigate some alternatives for Multics. Finally, in 1969, they withdrew their efforts from Multics, just like the other partners did. Ken Thompson wanted to create a comfortable computer environment constructed according to his own designs, of course, incorporating many of the innovative aspects of Multics. The group also left the idea of implementing solely in a higher-level language, allthough they also used BCPL for some portions. At that time they did not put much weight on portability. In 1968, Ken Thompson wrote the first, still very primitive Unix kernel, mainly in assembler, on the DEC PDP-7 computer with only 8kb RAM.
1969	TMG for Unix Doug McIlroy (Bell Labs) implemented the first high-level compiler for Unix.
1969	B for Unix Ken Thompson decided that Unix also needed a system programming language. First he considered to implement Fortran but then he created his own typeless language B, which was in fact a very squeezed version of BCPL to fit in the 8kB. After finishing, he wrote B in itself (the bootstrapping technique). During development, he continually struggled against memory limitations, but each rewrite taking advantage of some new features

	reduced its size. He introduced the arithmetic assignment operators from Algol (+=, -+,... which were at that time written as =+, -=,...) and invented the pre-and post/in-and decrement operators. Except B itself, no other programs were ever written in that language., because the machine was too small and too slow to do more than an experiment.
1970	They acquired the new DEC PDP-11 with 24 kB RAM and Ken Thompson recoded the Unix kernel in PDP-11 assembly language. They also ported B to their new computer.
1971	More and more programs were writtin in B on that new machine, for example Steve Johnson's yacc. However, the changed hardware word length (16-bit) and the promised floating-point operations in the PDP-11 exposed several inadequacies of B's semantic model.
1971	C Dennis Ritchie expanded the language with types (`int` and `char`, with arrays and pointers to these types) and changed the name to C. The intermediate name for the new language was NB (New B).
1972-1973	The C languages was further expanded. A very imported addition was the introduction of the preprocessor. It was based on the PL/I preprocessor, modified according to some ideas from Alan Snyder. At first, it was considered optional and not used very often.
1973	The essentials of modern C were complete. The Unix kernel was completely rewritten using C by Ken Thompson and Dennis Ritchie. Also, the C language was also ported to other platforms, including the IBM 360. To enable this, Mike Lesk wrote a portable I/O library which later became the standard C I/O library.
1973-1978	Further expansion of the C language: * more types and a more safety type system in general * portability issues Steven Johnson worked on a C compiler that was easy to retarget to new machines. Furthermore, Steve Johnson, Ken Thompson and Dennis Ritchie started to port Unix itself to other platforms (which also demonstrated the portability of C). Tom London and John Reiser ported Unix and C to the DEC VAX, which became a very popular machines, especially in the academic world. As a result, the use of Unix (and C) grew considerably during that period.
1978	"The C Programming language", written by Brian Kernighan and Dennis Ritchie.
1980	The C language also got popular outside the Unix world and even became the main programming language of choice in the IBM-PC world.
1983-1989	ANSI C standardization Doug McIlroy suggested ANSI to prepare a C standard. The standard contained one very important change to the language itself: the type of the formal function arguments was included in the signature of the function (modelled after C++).

Language overview

BCPL, B and C differ syntactically in many details but broadly they are similar and all belong to the procedural programming language family such as Fortran, Algol and Pascal. They are all 'close to the machine' in that their abstractions are founded on the concrete datatypes of the hardware, what makes them perfectly suited for system programming, but at the same time providing enough high-level abstractions.

	BCPL	B	C	Modelled after
Comments	// ... /* ... */	/* ... */	/* ... */	PL/I
Declaration syntax		begins with `auto` or `static`	including datatype	Fortran
Typed	typeless	typeless	strongly typed	Algol
Keywords	`switchon` `endcase`	`switch` `break`	`switch` `break`	
Assignment	`:=`	=	=	Fortran
Address	@	&	&	
Indirection	`rv or !`	*	*	Assembler (IBM 709)
Member	NA	NA	.	Cobol
Dereferencing	NA	NA	->	PL/I
Increment	NA	++ (pre and post)	++	Cobol (increment)
Arithmetic assignment	NA	=+	+=	Algol 68
Code	`let V = vec 10` `V!i`	`auto V[10];` `V[i]`		

In general, BCPL allows the nesting of procedures, procedure and data declaration have a more uniform structure and a more complete set of loops is available.
Both BCPL and B were typeless, or correcter, there was only one datatype: the memory cell. Memory was thus seen as a large linear array of elementary cells, the meaning of the content depended on the operator applied. This linear array view on memory allowed the interpretation of the value in a cell as an index in this array: pointers. This justifies the introduction of the rv operator and pointer arithmetic in BCPL.

One of the most significant changes from B to C, except for the addition of datatypes, was the relationship between arrays and pointers. C eliminated the materialization of the pointer in storage, and instead caused the creation of the pointer when the array name is mentioned in an expression. This convention enabled the easy storage of an array that is contained in a structure (without the extra storage of the pointer to the name).

The concepts of unions and casting were mainly captured from Algol. At first, the added preprocessor only recognized the inclusion of files (`#include`) and string replacements (`#define` of parameterless macros). Soon, it was extended by Mike Lesk and John Reiser to incorporate macros with parameters and conditional compilation.

Language examples

BCPL program:

```
// FACTORIAL
GET "LIBHDR"
LET START () BE $(
  LET F(N) = N=0 -> 1, N*F(N-1)
  FOR I = 1 TO 10 DO WRITEF("F(%N), = %N*N", I, F(I))
  FINISH
$)
```

Old-style C translation:

```
/* Factorial */
#include <stdio.h>
static f(n) /* in the BCPL version, this was local to main */
{
   return n == 0 ? 1 : n*f(n-1);
}
main()
{
  auto i;
  for (i = 1; i <= 10; i++)
    printf("f(%d), = %d\n", i, f(i));
  return 0;
}
```

BCPL program:

```
MANIFEST ${ TOPFACT = 10 $)
LET infact (n) = VALOF
$(
  LET f, j = 1., 0.

  FOR i = 0 TO n     // Declares i for the next block only
  $(
    f #*:= j;     // := is assign, = is compare
    fact!i := f;  // assignment doesn't return a value
    j #+:= 1.
  $)
  RESULTIS f
$)
AND fact = VEC TOPFACT;       // As in B, allocates 0 to TOPFACT
```

B equivalent:

```
infact (n)
{
  auto f, i, j;   /* no initialization for auto variables */
  extern fact;
  f = 1.;         /* floating point constant */
  j = 0.;
  for (i = 0; i <= n; ++i) {
    fact[i] = f =#* j;      /* note spelling =#* not #*= */
    j =#+ 1.;               /* #+ for floating add */
  }
  return (f);     /* at least, I think the () were required */
}
```

```
TOPFACT = 10;     /* equivalent of #define, numeric values only */
fact[TOPFACT];
```

C equivalent:

```
float infact (n) int n;
/* or, of course, the newer float infact (int n) */
{
  float f = 1;
  int i;
  extern float fact[];

  for (i = 0; i <= n; ++i)
    fact[i] = f *= i;
  return d;
}
#define TOPFACT 10
float fact[TOPFACT+1];
```

Original documents

- “The BCPL Reference Manual”, Martin Richards, MIT, 1967
- “The C Programming Language”, Brian Kernighan and Dennis Ritchie, Prentice Hall, 1978
- “The C Programming Language”, second edition, Brian Kernighan and Dennis Ritchie, Prentice Hall, 1988
- “TMG: A syntax directed compiler”, Doug McClure, 1965
- “The Programming Language B”, Steve Johnson and Brian Kernighan, 1973

Acknowledgement

The author would like to thank Doug McIlroy for his comments on the first version of this document.