History Is Written by the Winners

How C Programming Language Was Created And What Was Forgotten As a Result

Setting
(end 1960s — beginning 1970s)

- Mainframes
  - Advanced hardware
    - 24-bit to 31-bit addressing
    - Intelligent peripherals
  - Advanced software
    - Hypervisor (virtual machines)
    - Powerful programming languages
  - Extremely expensive
- Minicomputers
  - More affordable
  - Less powerful
  - More limited software

PDP-7

- Introduced in 1964
- 4K 18-bit words (9kB in modern units)
- Minimal addressable unit is **word**
- Unics OS: 1969
- B Programming language: 1969

The B Programming Language

- Single data type: 36-bit word

```
auto n;
```

- No floating point arithmetic
- Arrays as pointers

```
auto a[10];
```

- Example program:

```
main( ) {
  extrn a, b, c;
  putchar(a); putchar(b); putchar(c); putchar(’!*n’);
}
a ’hell’;
b ’o, w’;
c ’orld’;
```

PDP-11

- Introduced in 1970
- 16-bit address, 8-bit byte
- Orthogonal instruction set
- MMU
- Unibus / Q-bus
- Widely cloned in Soviet Union
- Porting UNIX begun in 1970

The C Programming Language

- Based on B, goal to preserve compatibility
- By 1973:
  - `char` datatype, typed arrays and pointers, real arrays that decay to pointers
  - Expression syntax for declarations

    ```
    int *api[10], (*pai)[10];
    ```
  - Structures (but with single namespace for members)
  - C preprocessor
- By 1977:
  - More type safety
  - Unsigned types
  - Casts
  - Each structure gets its own namespace for members
- By 1989: ANSI C

Meanwhile in other reality

- Algol 68 in 1968 (revised report in 1973)
- CLU in 1975
- Ada in 1983

Algol 68

- Introduced in 1968, revised report in 1973
- Everything is expression

```
int x := if a > b then a else b fi;
```

```
int y := (a > b | a | b)
```

- Variables are references, automatic dereferencing

```
int x;
```

```
ref int y = local int
```

- First class functions
- Advanced operator overloading
- Example program:

```
begin # Hello World in Algol 68 #
  print(("Hello World!", newline))
end
```

- Another example program:

```
(printf($"Hello, world!"l$)) ¢ Another Hello World in Algol 68 ¢
```

Ada

- Introduced in 1983
- Strong typing system
- Modules
- Builtin high-level multitasking
- Exceptions
- Generics
- Operator overloading
- Example program

```
-- Hello World in Ada
with Text_IO;
procedure Hello_World is
begin
  Text_IO.Put_Line("Hello World!");
end Hello_World;
```

CLU

- Introduced in 1975
- Clusters: abstract data types, almost classes
- Parameterised clusters, almost templates
- Iterators
- Efficient exceptions suitable for normal control flow
- Type-safe variant types
- Automatic memory management
- Operator overloading, including assignment to array element

```
a[27] :=3

array[int]$store(a, 27, 3)
```

- Example program:

```
% Hello, world in CLU
start_up = proc ()
    po: stream := stream$primary_output ()
    stream$putl(po, "Hello, world!")
end start_up
```

Interlude: ADT vs OOP vs structures

- Abstract data type:
  - Hidden internal structure
  - Exposed methods
- Object:
  - Interface
  - Concrete implementations
  - Constructors creating concrete implementations
- Structure:
  - Data members
  - Internal namespace for data members

Interlude: Clusters in CLU

- Abstract data type (or template of one)
  - Explicitly declared internal representation
- Not a structure (no data members)
- Not an object (no inheritance)

Interlude: Clusters in CLU

```
set = cluster [t: type] is create, insert, delete, is_in, size, elements, equal, copy, copy1
                        where t has equal: proctype (t, t) returns (bool)

  rep = array[t]

  create = proc () returns (cvt)
    return (rep$new())
  end create

  insert = proc (s: cvt, v: t)
    if ~is_in(up(s), v) then rep$addh(s, v) end
  end insert

  elements = iter (s: cvt) yields (t)
    for v: t in rep$elements(s) do
      yield (v)
    end
  end elements

  copy = proc (s: cvt) returns (cvt) where t has copy: proctype (t) returns (t)
    return (rep$copy(s))
  end copy

  copy1 = proc (s: cvt) returns (cvt)
    return (rep$copy1(s))
  end copy1

end set
```

Things that were restored quickly

- Type checks for argument types (ANSI C)
- `void` type (before ANSI C) [Algol 68]
- References (C++, 1983) [Algol 68]
- Constants (C++, 1983) [Algol 68]
- Templates (C++ 2.0 update, 1991) [CLU]
- Exceptions (C++ 2.0 update, 1991) [Ada]
- `bool` type (C++ 2.0 update, 1991) [Algol 68]

First class functions (43 years)

- Algol 68

```
begin
  proc apply int = (ref [] int a, proc (int) int f):
    for i from ⌊a to ⌈a do a[i] := f(a[i]) od;

  [1:3]int a := (1, 2, 3);
  apply int(a, proc(int n)int:(n × n))
end
```

- C++11

```
void apply_int(std::vector<int> &a,
               const std::function<int(int)> &f)
{
    for (int &elem: a)
        elem = f(elem);
}

int main()
{
    std::vector<int> a{1, 2, 3};
    apply_int(a, [](int n)->int{return n * n;});
}
```

if statement with initialiser (49 years)

- Algol 68

```
if
  int a = read int;
  int b = read int;
  a ≠ b
then
  print("Values are not equal!", newline)
fi
```

- C++17

```
if (int a, b; std::cin >> a >> b, a != b) {
    std::cout << "Values are not equal!"
              << std::endl;
}
```

Concepts (45 years?)

- CLU (1975)

```
set = cluster [t: type] is copy, …
                        where t has equal: proctype (t, t) returns (bool)
  …
  copy = proc (s: cvt) returns (cvt) where t has copy: proctype (t) returns (t)
    return (rep$copy(s))
  end copy
  …
end set
```

- C++20?

```
template <class T> concept bool EqualityComparable() {
    return requires(T a, T b) {
        {a == b} -> Boolean;
        {a != b} -> Boolean;
    };
}

template <EquaityComparable T> class set {
    …
};
```

Modules (33 years?)

- Ada

```
package Foo is
  procedure F (n: Natural);
end Foo;

with Text_IO;
package body Foo is
  procedure F (n: Natural) is
    Text_IO.Put_Line(n);
  end F;
begin
  Text_IO.Put_Line("Module Foo initialised");
end Foo;
```

- C++20?

```
import std;
module Foo;
export void f(unsigned int n) {
    std::cout << n << std::endl;
}
```

High-level multitasking

- Ada

```
task Buffer is
  entry Insert(D: Natural);
  entry Take(D: out Natural);
end Buffer;

task body Buffer is
  Length: constant Natural := 10;
  B: array(0..Length-1) of Natural;
  In_Ptr, Out_Ptr: Natural := 0;
  Count: Natural := 0;
begin
  loop
    select
      when Count < Length =>
        accept Insert(D: Natural) do
          B(In_Ptr) := D; In_Ptr := (In_Ptr +1) mod Length; Count := Count + 1;
        end Insert;
    or
      when Count > 0 =>
        accept Take(D: out Natural) do
          D := B(Out_Ptr); Out_Ptr := (Out_Ptr +1) mod Length; Count := Count - 1;
        end Take;
    or
      terminate;
    end select;
  end loop;
end Buffer;
```

References

- The circuit less traveled

  Investigating some alternate histories of computing

  (talk at FOSDEM 2018): https://fosdem.org/2018/schedule/event/alternative_histories/
- *The Development of the C Language* by Dennis M. Ritchie: https://www.bell-labs.com/usr/dmr/www/chist.html

Questions?