Software development: a personal and historical view

John Cremona
University of Warwick

5 October 2017

Introduction

- Who am I? A pure mathematician, working in number theory, with a special interest in "explicit methods" to solve problems in number theory and the development of algorithms in number theory

Introduction

- Who am I? A pure mathematician, working in number theory, with a special interest in "explicit methods" to solve problems in number theory and the development of algorithms in number theory
- What am I going to talk about?

Introduction

- Who am I? A pure mathematician, working in number theory, with a special interest in "explicit methods" to solve problems in number theory and the development of algorithms in number theory
- What am I going to talk about?
  - A personalized history: all (or most of) the different software I have used over the years as an aid to research in number theory (or just for fun).

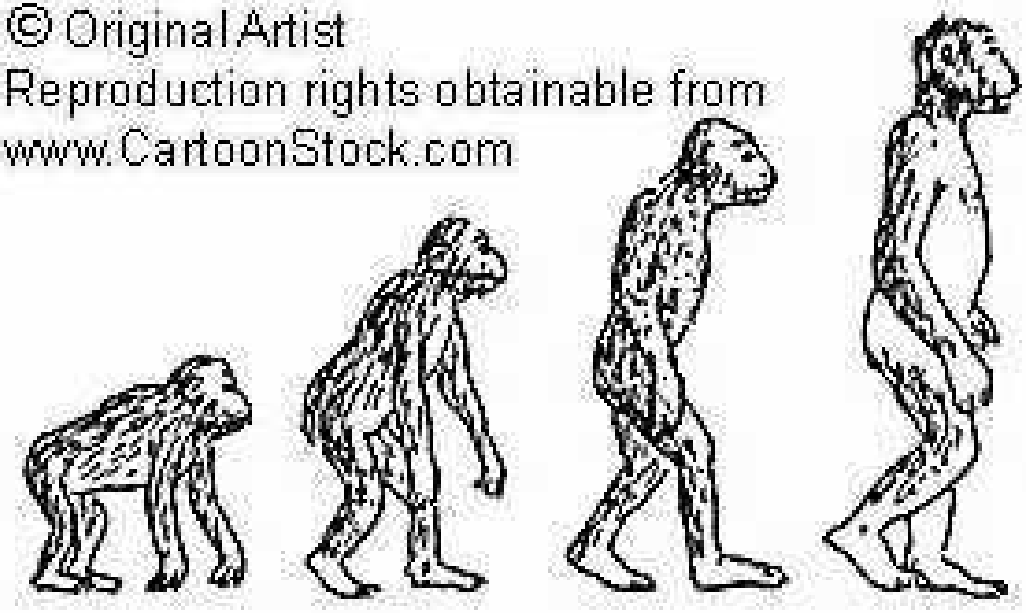
Introduction

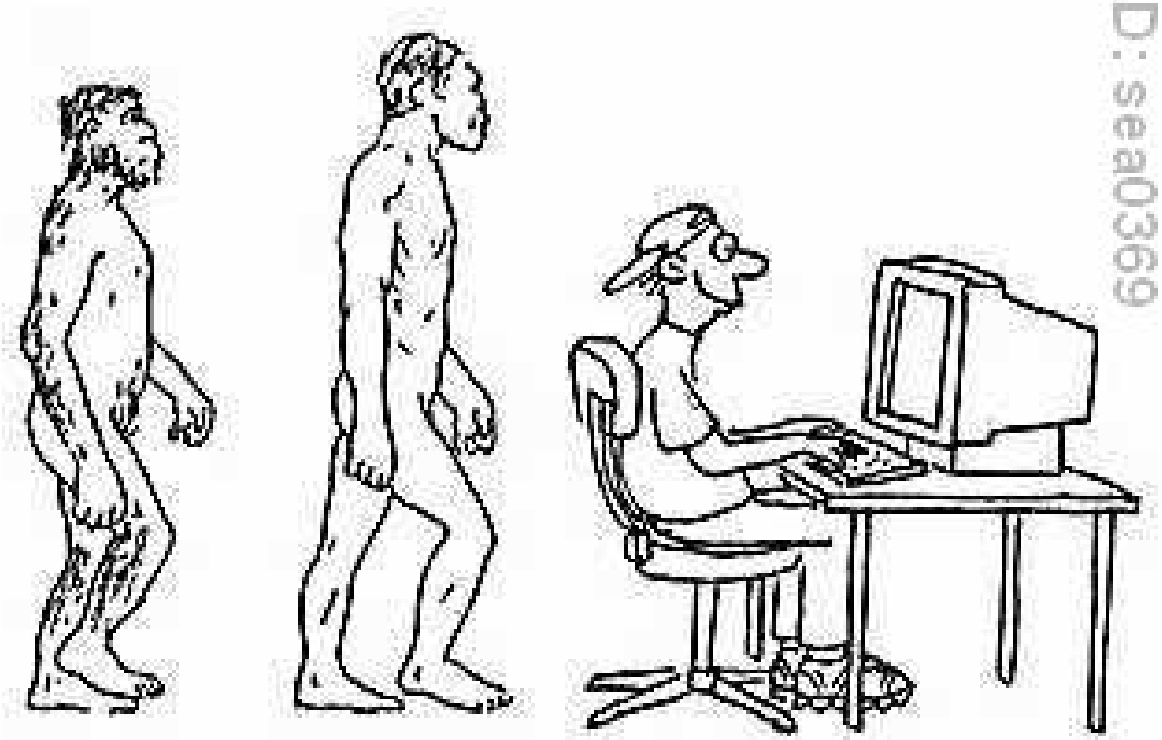
- Who am I? A pure mathematician, working in number theory, with a special interest in "explicit methods" to solve problems in number theory and the development of algorithms in number theory
- What am I going to talk about?
  - A personalized history: all (or most of) the different software I have used over the years as an aid to research in number theory (or just for fun).
  - An overview of how the mathematical software available to me has developed over the last 40 years: I will try to show how "you have never had it so good" — the software and related tools which we all have at our fingertips today, for free, enables us to be vastly more productive than I would have dreamed possible 40 years ago.

Introduction

- Who am I? A pure mathematician, working in number theory, with a special interest in "explicit methods" to solve problems in number theory and the development of algorithms in number theory
- What am I going to talk about?
  - A personalized history: all (or most of) the different software I have used over the years as an aid to research in number theory (or just for fun).
  - An overview of how the mathematical software available to me has developed over the last 40 years: I will try to show how "you have never had it so good" — the software and related tools which we all have at our fingertips today, for free, enables us to be vastly more productive than I would have dreamed possible 40 years ago.

Ask questions!





Prehistory I

- In around 1970 I was in high school, and a friend taught me `FORTRAN`.

Prehistory I

- In around 1970 I was in high school, and a friend taught me `FORTRAN`.
- My first ever program (which took days to write) was to list primes (using trial division).

Prehistory I

- In around 1970 I was in high school, and a friend taught me `FORTRAN`.
- My first ever program (which took days to write) was to list primes (using trial division).
- My second was to find gcds, using their prime factorizations (I had not heard of the Euclidean Algorithm). This one never worked properly.

Prehistory I

- In around 1970 I was in high school, and a friend taught me `FORTRAN`.
- My first ever program (which took days to write) was to list primes (using trial division).
- My second was to find gcds, using their prime factorizations (I had not heard of the Euclidean Algorithm). This one never worked properly.
- Our programs were typed onto paper tape. If you made a typo, or had to fix a bug, you had to read the tape in line by line, and either keep the line or delete it and/or insert a new one. This was tedious...

Prehistory I

- In around 1970 I was in high school, and a friend taught me `FORTRAN`.
- My first ever program (which took days to write) was to list primes (using trial division).
- My second was to find gcds, using their prime factorizations (I had not heard of the Euclidean Algorithm). This one never worked properly.
- Our programs were typed onto paper tape. If you made a typo, or had to fix a bug, you had to read the tape in line by line, and either keep the line or delete it and/or insert a new one. This was tedious...
- I got quite good at finding primes. Integers were 16-bit (up to 32767) but I got up to about 108,000. I kept the printout (the only output of the only run) on my wall for years.

Prehistory II

- My friend's father gave him access to a computer at the University of Cambridge, but only on Sunday evenings. The computer (an IBM1130) lived in a caravan behind the Observatory, and spent 6 months per year at sea (it belonged to the Department of Geodesy and Geophysics).

Prehistory II

- My friend's father gave him access to a computer at the University of Cambridge, but only on Sunday evenings. The computer (an IBM1130) lived in a caravan behind the Observatory, and spent 6 months per year at sea (it belonged to the Department of Geodesy and Geophysics).
- The computer was controlled from a teletype whose initial prompt said "Enter monitor control record. Job card group only." If what you responded was not right, it just repeated the prompt.

Prehistory II

- My friend's father gave him access to a computer at the University of Cambridge, but only on Sunday evenings. The computer (an IBM1130) lived in a caravan behind the Observatory, and spent 6 months per year at sea (it belonged to the Department of Geodesy and Geophysics).
- The computer was controlled from a teletype whose initial prompt said "Enter monitor control record. Job card group only." If what you responded was not right, it just repeated the prompt.
- If the machine crashed (which it did) rebooting it involved threading a special boot tape into it and setting a bank of around 30 toggle switches to some binary pattern, and then. . . I forget the rest.

Prehistory II

- My friend's father gave him access to a computer at the University of Cambridge, but only on Sunday evenings. The computer (an IBM1130) lived in a caravan behind the Observatory, and spent 6 months per year at sea (it belonged to the Department of Geodesy and Geophysics).
- The computer was controlled from a teletype whose initial prompt said "Enter monitor control record. Job card group only." If what you responded was not right, it just repeated the prompt.
- If the machine crashed (which it did) rebooting it involved threading a special boot tape into it and setting a bank of around 30 toggle switches to some binary pattern, and then. . . I forget the rest.
- It had a core storage: if you looked into the box you could see the tiny metal rings threaded onto wires, storing one bit each.

Software and training

- I used `FORTRAN` until I left high school. I had heard of Algol but it looked rather exotic, and its programs were all indented which looked strange.

Software and training

- I used `FORTRAN` until I left high school. I had heard of Algol but it looked rather exotic, and its programs were all indented which looked strange.
- At 17, I gained my first qualification in computing, an A Level in Computer Science (grade A!). It was not offered at my school but at a local college where I went in the evenings with 3 friends.

Software and training

- I used `FORTRAN` until I left high school. I had heard of Algol but it looked rather exotic, and its programs were all indented which looked strange.
- At 17, I gained my first qualification in computing, an A Level in Computer Science (grade A!). It was not offered at my school but at a local college where I went in the evenings with 3 friends.
- We had access to a more advanced 1130 which had card readers, so editing programs was easier. We had to queue outside the computer room with our card deck in hand until we were allowed in, one at a time, to feed our cards in. One syntax error and you went back to the card punches and back to the back of the queue.

Software and training

- I used `FORTRAN` until I left high school. I had heard of Algol but it looked rather exotic, and its programs were all indented which looked strange.
- At 17, I gained my first qualification in computing, an A Level in Computer Science (grade A!). It was not offered at my school but at a local college where I went in the evenings with 3 friends.
- We had access to a more advanced 1130 which had card readers, so editing programs was easier. We had to queue outside the computer room with our card deck in hand until we were allowed in, one at a time, to feed our cards in. One syntax error and you went back to the card punches and back to the back of the queue.
- That is my *only* qualification in computing.

High school projects

- As I was learning about Fourier series at school, I worked out the expansions of a few functions, and learned how to use the plotter, so I could plot graphs of the first few truncated Fourier polynomials for e^x.

High school projects

- As I was learning about Fourier series at school, I worked out the expansions of a few functions, and learned how to use the plotter, so I could plot graphs of the first few truncated Fourier polynomials for e^x.
- Exercise: do this in `SageMath` in the next five minutes!

Professional computing

- For six months during my "gap year" between school and university (and again the following summer) I worked at Addenbrookes Hospital, Cambridge as part of the Haematology Computer Project.

Professional computing

- For six months during my "gap year" between school and university (and again the following summer) I worked at Addenbrookes Hospital, Cambridge as part of the Haematology Computer Project.
- For a whole year, every blood sample analyzed at the hospital produced data which was dealt with by a program I wrote...

Professional computing

- For six months during my "gap year" between school and university (and again the following summer) I worked at Addenbrookes Hospital, Cambridge as part of the Haematology Computer Project.
- For a whole year, every blood sample analyzed at the hospital produced data which was dealt with by a program I wrote...
- ...still in `FORTRAN`.

Professional computing

- For six months during my "gap year" between school and university (and again the following summer) I worked at Addenbrookes Hospital, Cambridge as part of the Haematology Computer Project.
- For a whole year, every blood sample analyzed at the hospital produced data which was dealt with by a program I wrote...
- ...still in `FORTRAN`.
- I also had to debug programs written by other people who had left. This involved spending hours staring at `FORTRAN` programs full of conditional GOTO statements. Not nice.

Student computing

- There were no computers available or programming courses for mathematics undergraduates at Oxford then (1975–1978). Or any computing courses at all (just Numerical Analysis).

Student computing

- There were no computers available or programming courses for mathematics undergraduates at Oxford then (1975–1978). Or any computing courses at all (just Numerical Analysis).
- In the summer of my first year as a graduate student (1979) I started programming seriously, using `Algol68`. My qualifying dissertation (September 1979) contains a table of dimensions of Bianchi cusp forms of weight 2 for $\mathbb{Q}(\sqrt{-1})$ of all levels of norm up to 100...

Student computing

- There were no computers available or programming courses for mathematics undergraduates at Oxford then (1975–1978). Or any computing courses at all (just Numerical Analysis).
- In the summer of my first year as a graduate student (1979) I started programming seriously, using `Algol68`. My qualifying dissertation (September 1979) contains a table of dimensions of Bianchi cusp forms of weight 2 for $\mathbb{Q}(\sqrt{-1})$ of all levels of norm up to 100. . . which were almost all wrong. But they let me through!

Student computing

- There were no computers available or programming courses for mathematics undergraduates at Oxford then (1975–1978). Or any computing courses at all (just Numerical Analysis).
- In the summer of my first year as a graduate student (1979) I started programming seriously, using `Algol68`. My qualifying dissertation (September 1979) contains a table of dimensions of Bianchi cusp forms of weight 2 for $\mathbb{Q}(\sqrt{-1})$ of all levels of norm up to 100. . . which were almost all wrong. But they let me through!
- Since no-one had computed any of these before (except for degree 1 prime levels) no-one actually knew they were wrong! (And the bugs were fixed later.)

Graduate student computing

- Editing was easier. We had CRT terminals in the Mathematical Institute, connecting (via a system called Gandalf) to an ICL mainframe across the road.

Graduate student computing

- Editing was easier. We had CRT terminals in the Mathematical Institute, connecting (via a system called Gandalf) to an ICL mainframe across the road.
- There were two terminals. If you got in early you might get to use the one which ran at 300 baud, otherwise the other one was only 100 baud. (That is *slow*!)

Graduate student computing

- Editing was easier. We had CRT terminals in the Mathematical Institute, connecting (via a system called Gandalf) to an ICL mainframe across the road.
- There were two terminals. If you got in early you might get to use the one which ran at 300 baud, otherwise the other one was only 100 baud. (That is *slow*!)
- The editors provided on the system were useless. The one we all used (called "RAID") was written by a PhD student in mathematics, and was a lot better, though not a full-screen editor: you only saw one line of your program at a time.

Graduate student computing

- Editing was easier. We had CRT terminals in the Mathematical Institute, connecting (via a system called Gandalf) to an ICL mainframe across the road.
- There were two terminals. If you got in early you might get to use the one which ran at 300 baud, otherwise the other one was only 100 baud. (That is *slow*!)
- The editors provided on the system were useless. The one we all used (called "RAID") was written by a PhD student in mathematics, and was a lot better, though not a full-screen editor: you only saw one line of your program at a time. He never completed his PhD, but the editor was quite good for the time.

Graduate student computing

- Editing was easier. We had CRT terminals in the Mathematical Institute, connecting (via a system called Gandalf) to an ICL mainframe across the road.
- There were two terminals. If you got in early you might get to use the one which ran at 300 baud, otherwise the other one was only 100 baud. (That is *slow*!)
- The editors provided on the system were useless. The one we all used (called "RAID") was written by a PhD student in mathematics, and was a lot better, though not a full-screen editor: you only saw one line of your program at a time. He never completed his PhD, but the editor was quite good for the time.
- The mathematics graduate students who computed were split between number theorists (Birch students) and graph theorists (Cameron students), who used `Algol68`, and the rest (mostly applied mathematicians) who used `FORTRAN`.

Software reuse?

- My guru for both `Algol68` and the editor was Richard Pinch. He wrote both a multi-precision integer package and a number field package in `Algol68`.

Software reuse?

- My guru for both `Algol68` and the editor was Richard Pinch. He wrote both a multi-precision integer package and a number field package in `Algol68`. (He did complete his PhD.)
- Pinch had also programmed Tate's Algorithm (for elliptic curves over $\mathbb{Q}$, in `FORTRAN`) which I used for my own implementation over imaginary quadratic fields.

Software reuse?

- My guru for both `Algol68` and the editor was Richard Pinch. He wrote both a multi-precision integer package and a number field package in `Algol68`. (He did complete his PhD.)
- Pinch had also programmed Tate's Algorithm (for elliptic curves over $\mathbb{Q}$, in `FORTRAN`) which I used for my own implementation over imaginary quadratic fields.
- When Richard read a draft of my 1992 book "Algorithms for Modular Elliptic Curves" he noticed that the variable names in my description of Tate's Algorithm we exactly those from his 1980 `FORTRAN` program.

Software reuse?

- My guru for both `Algol68` and the editor was Richard Pinch. He wrote both a multi-precision integer package and a number field package in `Algol68`. (He did complete his PhD.)
- Pinch had also programmed Tate's Algorithm (for elliptic curves over $\mathbb{Q}$, in `FORTRAN`) which I used for my own implementation over imaginary quadratic fields.
- When Richard read a draft of my 1992 book "Algorithms for Modular Elliptic Curves" he noticed that the variable names in my description of Tate's Algorithm we exactly those from his 1980 `FORTRAN` program. And the same variable names are used in both `SageMath`'s and `Magma`'s implementations – which gives you a clue as to who wrote them!

The USA: Algol-free zone

- I spent 1981-2 at the University of Michigan, Ann Arbor.

The USA: Algol-free zone

- I spent 1981-2 at the University of Michigan, Ann Arbor. The Mathematics department had one computer, an Apple II (with *two* $5\frac{1}{4}$-inch floppy disk drives: one for the operating system and one for your programs and data). It ran `Pascal`.

The USA: Algol-free zone

- I spent 1981-2 at the University of Michigan, Ann Arbor. The Mathematics department had one computer, an Apple II (with *two* $5\frac{1}{4}$-inch floppy disk drives: one for the operating system and one for your programs and data). It ran `Pascal`.
- It was there that for the first time ever I computed (by hand) modular symbols for $\Gamma_0(11)$ and also (by hand) several Hecke eigenvalues, and hence (using the Apple II) the periods $(1.2692, 0.6346 + 1.4588i)$ and $c_4 = 496$, $c_6 = 20008$ of the "first" elliptic curve `11a1`.

The USA: Algol-free zone

- I spent 1981-2 at the University of Michigan, Ann Arbor. The Mathematics department had one computer, an Apple II (with *two* $5\frac{1}{4}$-inch floppy disk drives: one for the operating system and one for your programs and data). It ran `Pascal`.
- It was there that for the first time ever I computed (by hand) modular symbols for $\Gamma_0(11)$ and also (by hand) several Hecke eigenvalues, and hence (using the Apple II) the periods $(1.2692, 0.6346 + 1.4588i)$ and $c_4 = 496$, $c_6 = 20008$ of the "first" elliptic curve `11a1`.
- Programming the Apple II in `Pascal` was so painful (there were no built-in complex numbers) that I did most of it by hand with a pocket calculator.

Pascal and Macsyma

- In 1982-84 I was at Dartmouth College, the home and birthplace of BASIC.

Pascal and Macsyma

- In 1982-84 I was at Dartmouth College, the home and birthplace of BASIC.
- So I developed my Pascal skills. For a project on Abelian Varieties I used a mixture of Pascal and Macsyma.

Pascal and Macsyma

- In 1982-84 I was at Dartmouth College, the home and birthplace of BASIC.
- So I developed my Pascal skills. For a project on Abelian Varieties I used a mixture of Pascal and Macsyma.
- BASIC was only used for teaching: all students had to learn to code in it, and to swim, before they could graduate.

Pascal and Macsyma

- In 1982-84 I was at Dartmouth College, the home and birthplace of BASIC.
- So I developed my Pascal skills. For a project on Abelian Varieties I used a mixture of Pascal and Macsyma.
- BASIC was only used for teaching: all students had to learn to code in it, and to swim, before they could graduate.
- Macsyma was then an experimental program being developed at MIT; Dartmouth had a Developers' License, which meant it was free on condition that you reported bugs.

Pascal and Macsyma

- In 1982-84 I was at Dartmouth College, the home and birthplace of BASIC.
- So I developed my Pascal skills. For a project on Abelian Varieties I used a mixture of Pascal and Macsyma.
- BASIC was only used for teaching: all students had to learn to code in it, and to swim, before they could graduate.
- Macsyma was then an experimental program being developed at MIT; Dartmouth had a Developers' License, which meant it was free on condition that you reported bugs. Which I did.

Pascal and Macsyma

- In 1982-84 I was at Dartmouth College, the home and birthplace of BASIC.
- So I developed my Pascal skills. For a project on Abelian Varieties I used a mixture of Pascal and Macsyma.
- BASIC was only used for teaching: all students had to learn to code in it, and to swim, before they could graduate.
- Macsyma was then an experimental program being developed at MIT; Dartmouth had a Developers' License, which meant it was free on condition that you reported bugs. Which I did.
- The machines which ran Pascal and Macsyma were different. I computed the matrices of Hecke operators in Pascal on one machine, and used some kind of ftp to transfer files to the other machine where I used Macsyma to find their eigenvalues.

UK and Algol68 again

- In 1985 I moved back to the UK, and faced the problem of transferring my Pascal programs from Dartmouth and my old Algol68 programs from Oxford.

UK and Algol68 again

- In 1985 I moved back to the UK, and faced the problem of transferring my Pascal programs from Dartmouth and my old Algol68 programs from Oxford.
- Luckily at Exeter I had access to a large ICL mainframe at SWURCC (South West Universities Regional Computer Centre) similar to the one at Oxford, and with the same Algol68 compiler.

UK and Algol68 again

- In 1985 I moved back to the UK, and faced the problem of transferring my Pascal programs from Dartmouth and my old Algol68 programs from Oxford.
- Luckily at Exeter I had access to a large ICL mainframe at SWURCC (South West Universities Regional Computer Centre) similar to the one at Oxford, and with the same Algol68 compiler.
- I was able to transfer my files from Oxford (by writing a letter to OUCS asking them to copy them onto a magnetic tape which they posted and which the Exeter Computer Centre could read for me).

UK and Algol68 again

- In 1985 I moved back to the UK, and faced the problem of transferring my Pascal programs from Dartmouth and my old Algol68 programs from Oxford.
- Luckily at Exeter I had access to a large ICL mainframe at SWURCC (South West Universities Regional Computer Centre) similar to the one at Oxford, and with the same Algol68 compiler.
- I was able to transfer my files from Oxford (by writing a letter to OUCS asking them to copy them onto a magnetic tape which they posted and which the Exeter Computer Centre could read for me).
- But all Dartmouth could do was to print out all my files with a line-printer and mail the hard copy to me in a large parcel!

UK and Algol68 again

- In 1985 I moved back to the UK, and faced the problem of transferring my Pascal programs from Dartmouth and my old Algol68 programs from Oxford.
- Luckily at Exeter I had access to a large ICL mainframe at SWURCC (South West Universities Regional Computer Centre) similar to the one at Oxford, and with the same Algol68 compiler.
- I was able to transfer my files from Oxford (by writing a letter to OUCS asking them to copy them onto a magnetic tape which they posted and which the Exeter Computer Centre could read for me).
- But all Dartmouth could do was to print out all my files with a line-printer and mail the hard copy to me in a large parcel!
- I never used Pascal again.

From Algol68 to C++

- I continued using Algol68 for about ten years.

From Algol68 to C++

- I continued using Algol68 for about ten years.
- Certainly, all the data in my 1992 book (containing tables of elliptic curves of conductor ≤ 1000) was computed at SWURCC with Algol68.

From Algol68 to C++

- I continued using Algol68 for about ten years.
- Certainly, all the data in my 1992 book (containing tables of elliptic curves of conductor ≤ 1000) was computed at SWURCC with Algol68.
- What changed?

From Algol68 to C++

- I continued using Algol68 for about ten years.
- Certainly, all the data in my 1992 book (containing tables of elliptic curves of conductor ≤ 1000) was computed at SWURCC with Algol68.
- What changed?
  - SWURCC was going to be shut down (budget cuts and the rise of new machines such as Sun workstations and IBM PCs).

From Algol68 to C++

- I continued using Algol68 for about ten years.
- Certainly, all the data in my 1992 book (containing tables of elliptic curves of conductor ≤ 1000) was computed at SWURCC with Algol68.
- What changed?
  - SWURCC was going to be shut down (budget cuts and the rise of new machines such as Sun workstations and IBM PCs).
  - People were asking for copies of my programs; and not happy to be told they were in Algol68!

From Algol68 to C++

- I continued using Algol68 for about ten years.
- Certainly, all the data in my 1992 book (containing tables of elliptic curves of conductor ≤ 1000) was computed at SWURCC with Algol68.
- What changed?
  - SWURCC was going to be shut down (budget cuts and the rise of new machines such as Sun workstations and IBM PCs).
  - People were asking for copies of my programs; and not happy to be told they were in Algol68!
  - I wanted to get my hands on the Sun machines (bought by Applied colleagues with grants – they were expensive!) which did not run Algol68, only FORTRAN; and C.

From Algol68 to C++

- I continued using Algol68 for about ten years.
- Certainly, all the data in my 1992 book (containing tables of elliptic curves of conductor ≤ 1000) was computed at SWURCC with Algol68.
- What changed?
  - SWURCC was going to be shut down (budget cuts and the rise of new machines such as Sun workstations and IBM PCs).
  - People were asking for copies of my programs; and not happy to be told they were in Algol68!
  - I wanted to get my hands on the Sun machines (bought by Applied colleagues with grants – they were expensive!) which did not run Algol68, only FORTRAN; and C.
  - I discovered C++.

Why C++?

- To me, moving to C looked like a move backwards; I was used to having user-defined classes, and overloaded operators, for example. I want to be able to write

$$a = b + c$$

  and not

$$\texttt{xyz\_add}(a, b, c)$$

  (where a, b, c may be complex numbers or vectors or matrices or points on an elliptic curves, or whatever).
- I wanted to use a mainstream language which others used:
  - –so that others could use the programs I wrote
  - –so that I could make use of libraries written by others

LiDIA, NTL and PARI

- I was influenced by the existence of Buchmann's LiDIA project (started early 1990s in Saarbruecken): they wrote a large amount of C++ code for number theory, in a library, well-written and easy to use.

LiDIA, NTL and PARI

- I was influenced by the existence of Buchmann's LiDIA project (started early 1990s in Saarbruecken): they wrote a large amount of C++ code for number theory, in a library, well-written and easy to use.
  Buchmann is both an excellent computational number theorist and a computer scientist, who assured me that C++ code was as efficient as C.

LiDIA, NTL and PARI

- I was influenced by the existence of Buchmann's LiDIA project (started early 1990s in Saarbruecken): they wrote a large amount of C++ code for number theory, in a library, well-written and easy to use.
  Buchmann is both an excellent computational number theorist and a computer scientist, who assured me that C++ code was as efficient as C.
- LiDIA was easier to use (as a library) than PARI, mainly because it was written in C++, no C.

LiDIA, NTL and PARI

- I was influenced by the existence of Buchmann's LiDIA project (started early 1990s in Saarbruecken): they wrote a large amount of C++ code for number theory, in a library, well-written and easy to use.
  Buchmann is both an excellent computational number theorist and a computer scientist, who assured me that C++ code was as efficient as C.
- LiDIA was easier to use (as a library) than PARI, mainly because it was written in C++, no C.
- I mostly used LiDIA as a front-end to gmp.

LiDIA, NTL and PARI

- I was influenced by the existence of Buchmann's LiDIA project (started early 1990s in Saarbruecken): they wrote a large amount of C++ code for number theory, in a library, well-written and easy to use.
  Buchmann is both an excellent computational number theorist and a computer scientist, who assured me that C++ code was as efficient as C.
- LiDIA was easier to use (as a library) than PARI, mainly because it was written in C++, no C.
- I mostly used LiDIA as a front-end to gmp.
- When I learned of Victor Shoup's NTL library I used it as an alternative to LiDIA: it was very much easier to install, and provided all the functionality I needed at the time.

LiDIA, NTL and PARI

- I was influenced by the existence of Buchmann's LiDIA project (started early 1990s in Saarbruecken): they wrote a large amount of C++ code for number theory, in a library, well-written and easy to use.
  Buchmann is both an excellent computational number theorist and a computer scientist, who assured me that C++ code was as efficient as C.
- LiDIA was easier to use (as a library) than PARI, mainly because it was written in C++, no C.
- I mostly used LiDIA as a front-end to gmp.
- When I learned of Victor Shoup's NTL library I used it as an alternative to LiDIA: it was very much easier to install, and provided all the functionality I needed at the time.
- My elliptic curve package eclib now uses NTL; for a long time it could be configured to use either NTL or LiDIA, as the code used a common interface to both.

LiDIA, NTL and PARI

- I was influenced by the existence of Buchmann's LiDIA project (started early 1990s in Saarbruecken): they wrote a large amount of C++ code for number theory, in a library, well-written and easy to use.
  Buchmann is both an excellent computational number theorist and a computer scientist, who assured me that C++ code was as efficient as C.
- LiDIA was easier to use (as a library) than PARI, mainly because it was written in C++, no C.
- I mostly used LiDIA as a front-end to gmp.
- When I learned of Victor Shoup's NTL library I used it as an alternative to LiDIA: it was very much easier to install, and provided all the functionality I needed at the time.
- My elliptic curve package eclib now uses NTL; for a long time it could be configured to use either NTL or LiDIA, as the code used a common interface to both. (SageMath includes eclib and NTL, but not LiDIA).

LiDIA RIP

- I contributed elliptic curve code to LiDIA, but found that they did not have a good mechanism for people outside Buchmann's own research group to contribute code. It was common to get no response: the LiDIA group had some very competent people in it, but they were all on short-term contracts (graduate students and postdocs) whose main responsibility was (reasonably!) to their own work. And Buchmann himself was not involved at all in writing code; he was more of a figurehead.

LiDIA RIP

- I contributed elliptic curve code to LiDIA, but found that they did not have a good mechanism for people outside Buchmann's own research group to contribute code. It was common to get no response: the LiDIA group had some very competent people in it, but they were all on short-term contracts (graduate students and postdocs) whose main responsibility was (reasonably!) to their own work. And Buchmann himself was not involved at all in writing code; he was more of a figurehead.
- These (and the very late switch to a GPL-compatible license) may be why LiDIA has faded away.

LiDIA RIP

- I contributed elliptic curve code to LiDIA, but found that they did not have a good mechanism for people outside Buchmann's own research group to contribute code. It was common to get no response: the LiDIA group had some very competent people in it, but they were all on short-term contracts (graduate students and postdocs) whose main responsibility was (reasonably!) to their own work. And Buchmann himself was not involved at all in writing code; he was more of a figurehead.
- These (and the very late switch to a GPL-compatible license) may be why LiDIA has faded away.
- And there are now alternatives!

Pari/GP

- According to Henri Cohen (who is the original Pari developer along with Batut, Bernardi, and Olivier – "BaBeCoOl"), Pari[1] was first released in 1990, having existed since 1988.

[1] The "PA" stands for Pascal. The decision to switch to C was made on the recommendation of Joe Buhler.

[2] e.g. Don Zagier

Pari/GP

- According to Henri Cohen (who is the original Pari developer along with Batut, Bernardi, and Olivier – "BaBeCoOl"), Pari[1] was first released in 1990, having existed since 1988.
- As with LiDIA and NTL, Pari was written by active number theory researchers as a tool for their own work, only later becoming a "public" package.

[1] The "PA" stands for Pascal. The decision to switch to C was made on the recommendation of Joe Buhler.

[2] e.g. Don Zagier

Pari/GP

- According to Henri Cohen (who is the original Pari developer along with Batut, Bernardi, and Olivier – "BaBeCoOl"), Pari[1] was first released in 1990, having existed since 1988.
- As with LiDIA and NTL, Pari was written by active number theory researchers as a tool for their own work, only later becoming a "public" package.
- GP, the interactive command-line interface to Pari, is easy to powerful and use, and has played an enormously important role in computational and experimental number theory for the last 30 years.

[1] The "PA" stands for Pascal. The decision to switch to C was made on the recommendation of Joe Buhler.

[2] e.g. Don Zagier

Pari/GP

- According to Henri Cohen (who is the original Pari developer along with Batut, Bernardi, and Olivier – "BaBeCoOl"), Pari[1] was first released in 1990, having existed since 1988.
- As with LiDIA and NTL, Pari was written by active number theory researchers as a tool for their own work, only later becoming a "public" package.
- GP, the interactive command-line interface to Pari, is easy to powerful and use, and has played an enormously important role in computational and experimental number theory for the last 30 years.
- There are many useful GP scripts available on many mathematicians' web pages. These have varying quality, and are written in various different dialects of the GP language. Some mathematicians[2] use GP *a lot*, (and no other software), and still use a version now 20 years old!

[1] The "PA" stands for Pascal. The decision to switch to C was made on the recommendation of Joe Buhler.

[2] e.g. Don Zagier

Pari vs. GP

- Pari itself is a collection of C functions which form an extensive library which can be called from C/C++ programs.

Pari vs. GP

- Pari itself is a collection of C functions which form an extensive library which can be called from C/C++ programs.
- The interactive command-line interface GP was originally written as a platform for testing the "real" library functions.

Pari vs. GP

- Pari itself is a collection of C functions which form an extensive library which can be called from C/C++ programs.
- The interactive command-line interface GP was originally written as a platform for testing the "real" library functions.
- GP is a lot easier to use, and has far more users than the C library itself.

Pari vs. GP

- Pari itself is a collection of C functions which form an extensive library which can be called from C/C++ programs.
- The interactive command-line interface GP was originally written as a platform for testing the "real" library functions.
- GP is a lot easier to use, and has far more users than the C library itself.
- Since GP is interpreted, it is a lot slower than using the Pari library directly: but the human time needed to write code for the Pari library used to more than make up the difference!

Pari vs. GP

- Pari itself is a collection of C functions which form an extensive library which can be called from C/C++ programs.
- The interactive command-line interface GP was originally written as a platform for testing the "real" library functions.
- GP is a lot easier to use, and has far more users than the C library itself.
- Since GP is interpreted, it is a lot slower than using the Pari library directly: but the human time needed to write code for the Pari library used to more than make up the difference!
- Since 2002, Bill Allombert has developed a GP-to-C "compiler" called gp2c, which completely automates this process.

Pari vs. GP

- Pari itself is a collection of C functions which form an extensive library which can be called from C/C++ programs.
- The interactive command-line interface GP was originally written as a platform for testing the "real" library functions.
- GP is a lot easier to use, and has far more users than the C library itself.
- Since GP is interpreted, it is a lot slower than using the Pari library directly: but the human time needed to write code for the Pari library used to more than make up the difference!
- Since 2002, Bill Allombert has developed a GP-to-C "compiler" called gp2c, which completely automates this process.
  This is very similar to using Cython to speed up Python.

Pari development

- Pari started in Bordeaux in Henri Cohen's research group A^2X, and its development has remained centred there: now led by Karim Belabas and Bill Allombert.

Pari development

- Pari started in Bordeaux in Henri Cohen's research group A^2X, and its development has remained centred there: now led by Karim Belabas and Bill Allombert.
- It has a bug tracking system, and a growing test suite (which is added to whenever bugs are fixed).

Pari development

- Pari started in Bordeaux in Henri Cohen's research group A^2X, and its development has remained centred there: now led by Karim Belabas and Bill Allombert.
- It has a bug tracking system, and a growing test suite (which is added to whenever bugs are fixed).
- It has various mailing lists (pari-dev, pari-support, pari-announce) for a long time managed by a robot in Dan Bernstein's office.

Pari development

- Pari started in Bordeaux in Henri Cohen's research group A^2X, and its development has remained centred there: now led by Karim Belabas and Bill Allombert.
- It has a bug tracking system, and a growing test suite (which is added to whenever bugs are fixed).
- It has various mailing lists (pari-dev, pari-support, pari-announce) for a long time managed by a robot in Dan Bernstein's office.
- Relatively few people work on Pari development: mostly KB, BA and Jeroen Demeyer (from SageMath).

Pari development

- Pari started in Bordeaux in Henri Cohen's research group A^2X, and its development has remained centred there: now led by Karim Belabas and Bill Allombert.
- It has a bug tracking system, and a growing test suite (which is added to whenever bugs are fixed).
- It has various mailing lists (pari-dev, pari-support, pari-announce) for a long time managed by a robot in Dan Bernstein's office.
- Relatively few people work on Pari development: mostly KB, BA and Jeroen Demeyer (from SageMath).
- The heavy use of Pari in SageMath has revealed many bugs in Pari, all of which have been rapidly fixed upstream.

Pari and SageMath

- The relationship between Pari and SageMath is symbiotic, though not at all symmetrical.

Pari and SageMath

- The relationship between Pari and SageMath is symbiotic, though not at all symmetrical.
- SageMath relies on Pari entirely for all its number field functionality (with very few exceptions).

Pari and SageMath

- The relationship between Pari and SageMath is symbiotic, though not at all symmetrical.
- SageMath relies on Pari entirely for all its number field functionality (with very few exceptions).
- In turn, SageMath users greatly increase the Pari user base, and so SageMath (and all the relevant doctests) provides a great platform for testing Pari (finding bugs, occasionally fixing them, helping to improve the build system).

Magma

- Magma 1.0 was released in 1993, but it was preceded for 20 years by Cayley, a package which concentrated on group theory.

Magma

- Magma 1.0 was released in 1993, but it was preceded for 20 years by Cayley, a package which concentrated on group theory.
- Magma covers many areas of algebra (particularly group theory), algebraic geometry, algebraic combinatorics, as well as number theory.

Magma

- Magma 1.0 was released in 1993, but it was preceded for 20 years by Cayley, a package which concentrated on group theory.
- Magma covers many areas of algebra (particularly group theory), algebraic geometry, algebraic combinatorics, as well as number theory.
- An important reason why many number theorists prefer to use Magma is its excellent support for algebraic geometry.

Magma

- Magma 1.0 was released in 1993, but it was preceded for 20 years by Cayley, a package which concentrated on group theory.
- Magma covers many areas of algebra (particularly group theory), algebraic geometry, algebraic combinatorics, as well as number theory.
- An important reason why many number theorists prefer to use Magma is its excellent support for algebraic geometry.
- There are around 4000 citations of Magma listed on the Magma web pages.

Magma

- Magma 1.0 was released in 1993, but it was preceded for 20 years by Cayley, a package which concentrated on group theory.
- Magma covers many areas of algebra (particularly group theory), algebraic geometry, algebraic combinatorics, as well as number theory.
- An important reason why many number theorists prefer to use Magma is its excellent support for algebraic geometry.
- There are around 4000 citations of Magma listed on the Magma web pages. ($\geq$ 12 by me!).

Magma

- Magma 1.0 was released in 1993, but it was preceded for 20 years by Cayley, a package which concentrated on group theory.
- Magma covers many areas of algebra (particularly group theory), algebraic geometry, algebraic combinatorics, as well as number theory.
- An important reason why many number theorists prefer to use Magma is its excellent support for algebraic geometry.
- There are around 4000 citations of Magma listed on the Magma web pages. (≥ 12 by me!).
- Magma uses its own strongly-typed language, and is based on sound (pedantic!) abstract mathematical principles.

Magma

- Magma 1.0 was released in 1993, but it was preceded for 20 years by Cayley, a package which concentrated on group theory.
- Magma covers many areas of algebra (particularly group theory), algebraic geometry, algebraic combinatorics, as well as number theory.
- An important reason why many number theorists prefer to use Magma is its excellent support for algebraic geometry.
- There are around 4000 citations of Magma listed on the Magma web pages. (≥ 12 by me!).
- Magma uses its own strongly-typed language, and is based on sound (pedantic!) abstract mathematical principles.
- Magma is neither free nor open source. Its core is written in C (not available to users) while much higher functionality is written in Magma'a own language, and is distributed.

Magma

- Magma 1.0 was released in 1993, but it was preceded for 20 years by Cayley, a package which concentrated on group theory.
- Magma covers many areas of algebra (particularly group theory), algebraic geometry, algebraic combinatorics, as well as number theory.
- An important reason why many number theorists prefer to use Magma is its excellent support for algebraic geometry.
- There are around 4000 citations of Magma listed on the Magma web pages. (≥ 12 by me!).
- Magma uses its own strongly-typed language, and is based on sound (pedantic!) abstract mathematical principles.
- Magma is neither free nor open source. Its core is written in C (not available to users) while much higher functionality is written in Magma'a own language, and is distributed.

SageMath

[We'll have a full introduction to SageMath later in the term.]

SageMath

[We'll have a full introduction to SageMath later in the term.]

The story of how William Stein came to be frustrated with earlier systems he used (mainly C++ and Magma, for different reasons) has been told many times.

SageMath

[We'll have a full introduction to SageMath later in the term.]

The story of how William Stein came to be frustrated with earlier systems he used (mainly C++ and Magma, for different reasons) has been told many times.

SageMath consists both of a huge library of classes and functions written in Python and Cython, and also of a large collection of third-party packages, all open source and with appropriate licences, integrated together into a seamless whole which builds easily on any computer and will one day be able to do anything which any other package mentioned here can do.

SageMath

[We'll have a full introduction to SageMath later in the term.]

The story of how William Stein came to be frustrated with earlier systems he used (mainly C++ and Magma, for different reasons) has been told many times.

SageMath consists both of a huge library of classes and functions written in Python and Cython, and also of a large collection of third-party packages, all open source and with appropriate licences, integrated together into a seamless whole which builds easily on any computer and will one day be able to do anything which any other package mentioned here can do.

SageMath also has a free online version called CoCalc (used to be SageMathCloud). See http://cocalc.com.

Why is SageMath different?

- "Not re-inventing the wheel": using all the good, free code which already exists.

Why is SageMath different?

- "Not re-inventing the wheel": using all the good, free code which already exists.
- Often making this code much easier to build and use.

Why is SageMath different?

- "Not re-inventing the wheel": using all the good, free code which already exists.
- Often making this code much easier to build and use.
- Using a mainstream powerful language (Python) instead of creating its own.

Why is SageMath different?

- "Not re-inventing the wheel": using all the good, free code which already exists.
- Often making this code much easier to build and use.
- Using a mainstream powerful language (Python) instead of creating its own.
- Very easy experimentation and interactive use...

Why is SageMath different?

- "Not re-inventing the wheel": using all the good, free code which already exists.
- Often making this code much easier to build and use.
- Using a mainstream powerful language (Python) instead of creating its own.
- Very easy experimentation and interactive use...
- ...with the capability of accessing the speed of compiled code (Cython).

Why is SageMath different?

- "Not re-inventing the wheel": using all the good, free code which already exists.
- Often making this code much easier to build and use.
- Using a mainstream powerful language (Python) instead of creating its own.
- Very easy experimentation and interactive use...
- ...with the capability of accessing the speed of compiled code (Cython).
- Choice of command-line use (e.g. for use in scripts) or notebook interface (user-friendly).

Why is SageMath different?

- "Not re-inventing the wheel": using all the good, free code which already exists.
- Often making this code much easier to build and use.
- Using a mainstream powerful language (Python) instead of creating its own.
- Very easy experimentation and interactive use...
- ...with the capability of accessing the speed of compiled code (Cython).
- Choice of command-line use (e.g. for use in scripts) or notebook interface (user-friendly).
- Ability to run a server for users anywhere.

Why is SageMath different?

- "Not re-inventing the wheel": using all the good, free code which already exists.
- Often making this code much easier to build and use.
- Using a mainstream powerful language (Python) instead of creating its own.
- Very easy experimentation and interactive use...
- ...with the capability of accessing the speed of compiled code (Cython).
- Choice of command-line use (e.g. for use in scripts) or notebook interface (user-friendly).
- Ability to run a server for users anywhere.
- Large and active community of developers.

Why is SageMath different?

- "Not re-inventing the wheel": using all the good, free code which already exists.
- Often making this code much easier to build and use.
- Using a mainstream powerful language (Python) instead of creating its own.
- Very easy experimentation and interactive use...
- ...with the capability of accessing the speed of compiled code (Cython).
- Choice of command-line use (e.g. for use in scripts) or notebook interface (user-friendly).
- Ability to run a server for users anywhere.
- Large and active community of developers.
- Easy transition from user to developer.

Why is SageMath different?

- "Not re-inventing the wheel": using all the good, free code which already exists.
- Often making this code much easier to build and use.
- Using a mainstream powerful language (Python) instead of creating its own.
- Very easy experimentation and interactive use...
- ...with the capability of accessing the speed of compiled code (Cython).
- Choice of command-line use (e.g. for use in scripts) or notebook interface (user-friendly).
- Ability to run a server for users anywhere.
- Large and active community of developers.
- Easy transition from user to developer.
- **free!**

Why I like using SageMath

- All of the above!

Why I like using SageMath

- All of the above!
- It is more powerful (in terms of what is implemented) than anything I have ever used, except (in some areas) Magma.

Why I like using SageMath

- All of the above!
- It is more powerful (in terms of what is implemented) than anything I have ever used, except (in some areas) Magma.
- It is very easy to collaborate with students (e.g. by sharing worksheets).

Why I like using SageMath

- All of the above!
- It is more powerful (in terms of what is implemented) than anything I have ever used, except (in some areas) Magma.
- It is very easy to collaborate with students (e.g. by sharing worksheets).
- Interfacing with other programs and the operating system is very easy (thanks to Python).

Why I like using SageMath

- All of the above!
- It is more powerful (in terms of what is implemented) than anything I have ever used, except (in some areas) Magma.
- It is very easy to collaborate with students (e.g. by sharing worksheets).
- Interfacing with other programs and the operating system is very easy (thanks to `Python`).
  Example: I still use my `C++` programs (`eclib`) to find modular elliptic curves, but now the post-processing of the curves to build the database is done entirely within `SageMath` (isogenies, generators, integral points, BSD data, ...) – using `Magma` or `Pari` for some functions (e.g. Heegner Points) where needed.

Why I like using SageMath

- All of the above!
- It is more powerful (in terms of what is implemented) than anything I have ever used, except (in some areas) Magma.
- It is very easy to collaborate with students (e.g. by sharing worksheets).
- Interfacing with other programs and the operating system is very easy (thanks to `Python`).
  Example: I still use my `C++` programs (`eclib`) to find modular elliptic curves, but now the post-processing of the curves to build the database is done entirely within `SageMath` (isogenies, generators, integral points, BSD data, ...) – using `Magma` or `Pari` for some functions (e.g. Heegner Points) where needed.
- The feeling that the code I write will *last*: instead of rotting away, becoming harder and then impossible to use, it will remain available for ever (or until replaced by improved code).

Other systems and packages

Time has not permitted me to discuss all relevant software, even that used in number-theory.I have concentrated on the ones I know best. There are around 75 items listed at `http://www.numbertheory.org/ntw/N1.html`, including all those mentioned here, several which are components of `SageMath`, and more.

Other systems and packages

Time has not permitted me to discuss all relevant software, even that used in number-theory.I have concentrated on the ones I know best. There are around 75 items listed at `http://www.numbertheory.org/ntw/N1.html`, including all those mentioned here, several which are components of `SageMath`, and more.

I will finish by mentioning a few of these.

Other systems and packages (continued)

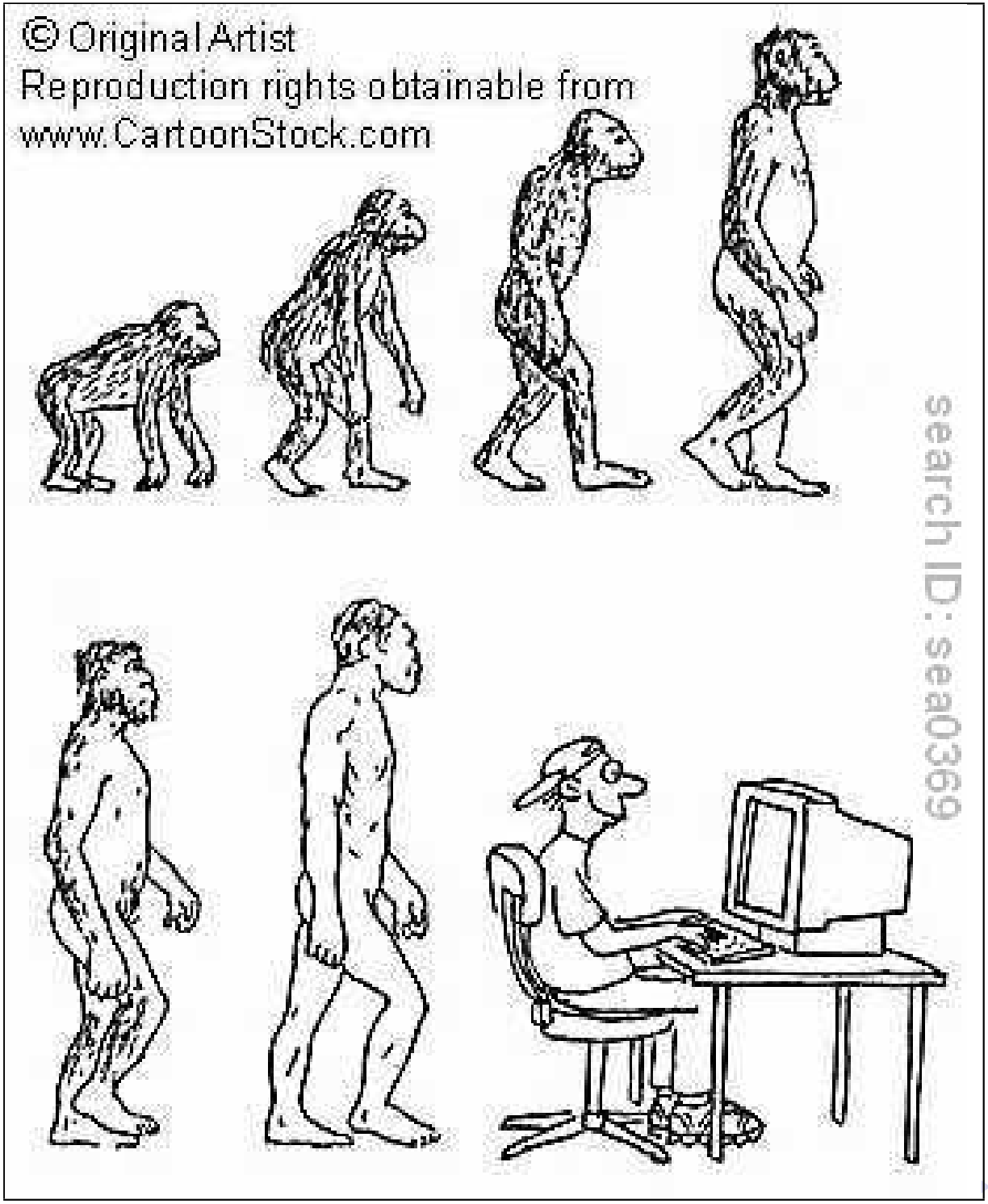
- Many other "low-level" multi-precision integer packages (`mpir`, `lip`, ...)

Other systems and packages (continued)

- Many other "low-level" multi-precision integer packages (`mpir`, `lip`, ...)
- The major "competitor" to `Pari` for computations with algebraic number fields used to be `KANT`, developed by Michael Pohst and his group; the `KANT` functionality was incorporated into `Magma`, and as far as I know, `KANT` is no longer very actively developed.

Other systems and packages (continued)

- Many other "low-level" multi-precision integer packages (`mpir`, `lip`, ...)
- The major "competitor" to `Pari` for computations with algebraic number fields used to be `KANT`, developed by Michael Pohst and his group; the `KANT` functionality was incorporated into `Magma`, and as far as I know, `KANT` is no longer very actively developed.
- `SiMATH`, developed by Horst Zimmer in Saarbruecken with the help of many of his students and assistants; funded by Siemens, its code was never open and is now lost, along with a vast amount of human effort.
- `Macaulay2` for algebraic geometry
- ...


© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com
search ID: sea0369