Systems Programming Languages

Advanced Systems Programming (H/M) 2025-2026 – Laboratory Exercise 1
School of Computing Science, University of Glasgow

1 Introduction

The Advanced Systems Programming (H/M) course uses the Rust programming language to illustrate several topics in systems programming. Before focussing on a new programming language, however, it's important to understand the strengths and limitations of the systems programming languages you already know. Accordingly, this exercise focusses on the C programming language.

This is a formative exercise and is not assessed.

2 Systems Programming in C

The Systems Programming (H) course, that many of you will have taken last year, introduced you to the concepts of systems programming using the C programming language. C has been one of the most widely used and successful programming languages for many years–the first standard for the C programming language was published in 1989–and remains the main implementation language for the Linux, Windows, macOS/iOS, and Android kernels and for large parts of their runtime support libraries.

Despite that success, C has a reputation of being difficult to learn and difficult to use securely. It sits at an awkward place in the spectrum of languages: it is sufficiently high-level to offer a reasonably portable abstraction of the machine while simultaneously operating at a low-enough level to provide important transparency in data layout and control over memory management. The abstraction of *pointers*, coupled with manual memory management, a weak type system, and the ability to set pointers to any location, and to cast between types to change how the contents of memory are interpreted, is essential to this.

As an example of the flexibility of C, consider the following (incomplete) fragment of C code, that might form part of the real-time transport protocol (RTP, RFC 3550 processing code for a voice-over-IP phone receiving data from a UDP socket, extracting the RTP packer header, parsing it, and then extracting the payload data for later processing:

```
struct rtp_header {
  uint16_t   cc:4
  uint16_t    x:1;
  uint16_t    p:1;
  uint16_t    v:2;
  uint16_t   pt:7;
  uint16_t    m:1;
  uint16_t  seq;
  uint32_t   ts;
  uint32_t ssrc;
}

...
char *buffer = malloc(BUFLEN);
int   fd = socket(...); // create UDP socket
...
```

```
int len = recv(fd, buffer, BUFLEN, 0);
if ((len > sizeof(struct rtp_header)) {
  struct rtp_header *rtp = (struct rtp_header *) buffer;
  if (rtp->v != 2) {
    // Error, wrong version number...
  } else {
    if (rtp->pt == 0) {
      struct pcm_payload *pcm = (struct pcm_payload *) (buffer + sizeof(struct rtp_header));
      ...
    } else if (rtp->pt == 3) {
      struct gsm_payload *gsm = (struct gsm_payload *) (buffer + sizeof(struct rtp_header));
      ...
    } else {
      ...
    }
  }
}
```

Discuss this code with the lecturer or lab demonstrator to ensure that you understand how it works and what it is supposed to do. Then, think about how you would implement a similar system in Python or Java, and how such a system would differ in how it accesses and/or copies memory and how it manipulates the various types of data.

Based on this example, and your experience writing C programs, think about the strengths and weaknesses of C, in terms of how it manages memory and handles pointers and types. Identify some strengths of C – areas where it offers useful features that do not exist in other languages – as well as some weaknesses. Individually, or in small groups, prepare two or three presentation slides to summarise your thinking. You'll discuss these slides during the lecture 2 discussion time next week.

This is a formative exercise and is not assessed.

The slides you prepare are intended to help start the discussion about needs of systems programming languages in the discussion time associated with Lecture 2, next week. They do not need to be submitted and will not be assessed, but do bring them to Lecture 2 you can present them and we can discuss the issues you raise.