C++ terms

History

- Bjarne Stroustrup:
  - Name pronunciation: https://youtu.be/9QKHg8wj4MA
  - Why C++?: https://youtu.be/JBjjnqG0BP8
    - C's low-level programming + OOP for ease of programming abstraction
- C++ versions: 11, 14, 17, etc.

Pointers

Raw pointers: `&` and `*`

Links:

- https://www.cplusplus.com/doc/tutorial/pointers/
- https://docs.microsoft.com/en-us/cpp/cpp/pointers-cpp?view=msvc-170

We will be using pointers quite extensively in this unit, especially when designing object oriented data structures. So make sure you understand and are comfortable with them.

- `&`: address-of operator, which precedes a variable name and gives the memory address of the variable
  - `foo = &var`: `foo` is called the **pointer** to `var`
- `*`: dereference operator (or 'value pointed to by'), which gives access to the value of the variable that the pointer is pointing to
  - `*foo` gives `var`'s value

Smart pointers

Link: https://docs.microsoft.com/en-us/cpp/cpp/smart-pointers-modern-cpp?view=msvc-170

Raw pointer usage is generally discouraged. In general, you should use smart pointer instead!

Pre-processor Directives

Links: https://www.cplusplus.com/doc/tutorial/preprocessor/

- Preprocessor directives are lines included in the code of programs preceded by a hash sign (#).
- Are not program statements but directives for the preprocessor
- The preprocessor examines the code before actual compilation of code begins and resolves all these directives before any code is actually generated by regular statements.
- By default, extend only across a single line of code:
  - can extend through more than one line by preceding the newline character at the end of the line by a backslash ()
- No semicolon (`;`) is expected at the end of a preprocessor directive.

Exammple:

```
#ifndef HEADER_H_
#define HEADER_H_
/* Body of Header */
#endif /* HEADER_H_ */
```

- `#define identifier replacement...#undef identifier`: macro definition: replaces any occurrence of `identifier` in the rest of the code by `replacement`
    - ended with `#undef identifier`
- `#ifdef identifier ...#endif`: conditional inclusion: include the code (`...`) only if `identifier` has been defined
- `#ifndef identifier ...#endif`: opposite of `ifdef`
- `#if...#elif...#else...`
- `#include ...`
- `#error`: abort compilation
- `#pragma...`: specifies options to compiler
    - `#pragma once`: the current file to be included at most once

C++ Assembly

How to view assembly code of the compiled program?

1. Use g++/gcc command
2. Use "Open Disassembler View" of the Debugger in VSCode

Command:

- `g++ -S MyProg.cpp`: produces `MyProg.s` for viewing.
- `gcc -save-temps -fverbose-asm MyProg.cpp`: produces comments and more info
- How to interpret the assembly code?
    - Link: https://vimeo.com/21720824
    - Havard: https://cs61.seas.harvard.edu/site/2021/Asm/

Example `Simple.cpp`:

```
    .file   "Simple.cpp"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
```

```
    .cfi_def_cfa_register 6
    movl    $0, %eax
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long    1f - 0f
    .long    4f - 1f
    .long    5
0:
    .string  "GNU"
1:
    .align 8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
3:
    .align 8
4:
```

Example: Simple2.cpp

```
    .file   "Simple2.cpp"
    .text
    .globl  _Z3sumRiS_
    .type   _Z3sumRiS_, @function
_Z3sumRiS_:
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movq    %rdi, -8(%rbp)
    movq    %rsi, -16(%rbp)
    movq    -8(%rbp), %rax
    movl    (%rax), %edx
    movq    -16(%rbp), %rax
    movl    (%rax), %eax
    addl    %edx, %eax
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
```

```
    .cfi_endproc
.LFE0:
    .size   _Z3sumRiS_, .-_Z3sumRiS_
    .globl  main
    .type   main, @function
main:
.LFB1:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $32, %rsp
    movq    %fs:40, %rax
    movq    %rax, -8(%rbp)
    xorl    %eax, %eax
    movl    $2, -20(%rbp)
    movl    $3, -16(%rbp)
    leaq    -16(%rbp), %rdx
    leaq    -20(%rbp), %rax
    movq    %rdx, %rsi
    movq    %rax, %rdi
    call    _Z3sumRiS_
    movl    %eax, -12(%rbp)
    movl    $0, %eax
    movq    -8(%rbp), %rcx
    xorq    %fs:40, %rcx
    je  .L5
    call    __stack_chk_fail@PLT
.L5:
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE1:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long    1f - 0f
    .long    4f - 1f
    .long    5
0:
    .string  "GNU"
1:
    .align 8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
3:
```

```
    .align 8
4:
```