Data, Code and Memory

hannes@ru.is

Declaration vs. Definition

- Declaration
  - Describes data object / function (name + type)
- Definition
  - Describes unique region of memory

Declaration vs. Definition

- Definitions
  - Two in SAME TRANS. UNIT → Compiler Error
  - Two in DIFFERENT TRANS. UNITS → Linker Error
- Therefore only declarations in header files!

- Also note:
  By default definitions have external linkage
  - Force internal with „static“

Memory Layout

- Executable file formats contain partial image of program
  - e.g. Executable and Linking Format ELF and EXE
- Layout in File
  - TEXT / CODE Segment
  - DATA Segment (initialized globals + statics)
  - BSS Segment (uninitialized globals + statics, dynamically filled with zero once loaded)
  - READ-ONLY DATA Segment (global constants)

Memory Layout

- Once loaded into RAM the layout remains, plus
  - BSS is expanded into full size and filled with zeros
  - PROGRAM STACK is added. Each function pushes a stack frame onto it when called, which contains:
    - RETURN ADDRESS
    - SAVED CPU REGISTERS
    - LOCAL VARIABLES
  - DYNAMIC ALLOCATION HEAP is added

Classes and Structs

- Class or Struct declarations allocate no memory
- Instances can be
  - Local variables on PROGRAM STACK
  - Global, file-static or function static in DATA/BSS
  - Dynamically allocated on the HEAP

Class Members

- Class static member
  - Not a regular member, instead acts as a global
- Class public vs. private members
  - Controls visibility

- NOTE:
  - Static at file scope → only seen in file (visibility)
  - Static at function scope → global but only seen in function (role + visibility)

Alignment and Packing

```
Struct mystruct {
    int         mID;
    char*       mName;
    bool        mAlive;
    int         mScore;
    char        mWeapon;
    float       mMultiplier;
}
```

4+4+1+4+1+4 = 18 bytes of data

BUT

Takes 24 bytes in memory!

Object Layout in Memory

- Compiler does not pack as tight as possible!
- The CPU does not at all read unaligned data (e.g. PS2) or may need to shift it around, losing performance
- **Alignment**
  - The address of an object needs to be a multiple of the data type size in bytes. E.g. a 32-bit integer needs to be at addresses that are divisible by 4

Alignment and Packing

```
Struct mystruct {
    int         mID;
    char*       mName;
    int         mScore;
    float       mMultiplier;
    bool        mAlive;
    char        mWeapon;
    char        _pad[2];
}
```

4+4+4+4+1+1 = 18 bytes of data

NOW occupies 20 bytes of memory, whereof 2 bytes are padding

Class Memory Layout

- Data members of classes appear together (aligned)
- Data members of derived classes appear after data members of parent classes
- Special data member for a virtual table pointer if virtual functions are used