CS601: Software Development for Scientific Computing

Autumn 2021

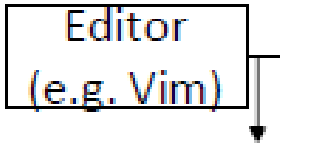
Week2: Program Development Environment, Minimal C++, Version Control Systems, Structured Grid

Program Development Environment

- Demo

Creating a Program

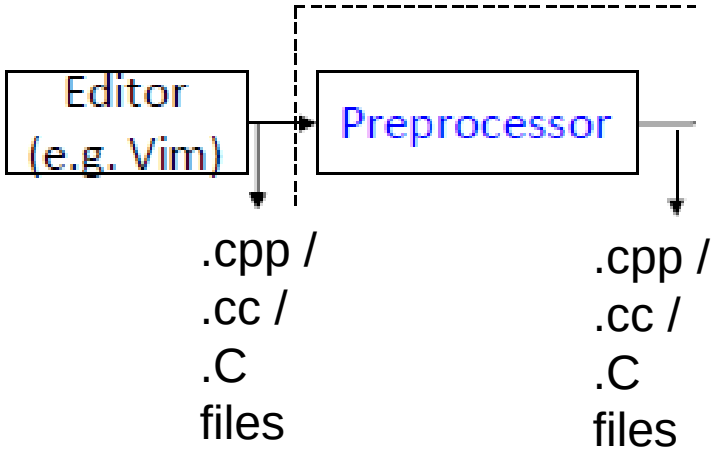
- Create your c++ program file



.cpp /
.cc /
.C
files

Creating a Program

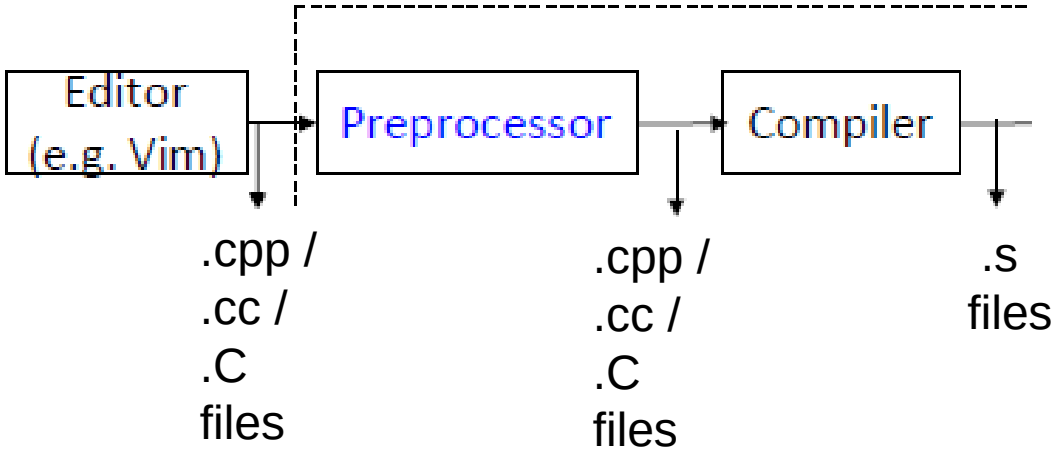
- Preprocess your c++ program file



- removes comments from your program,
- expands #include statements

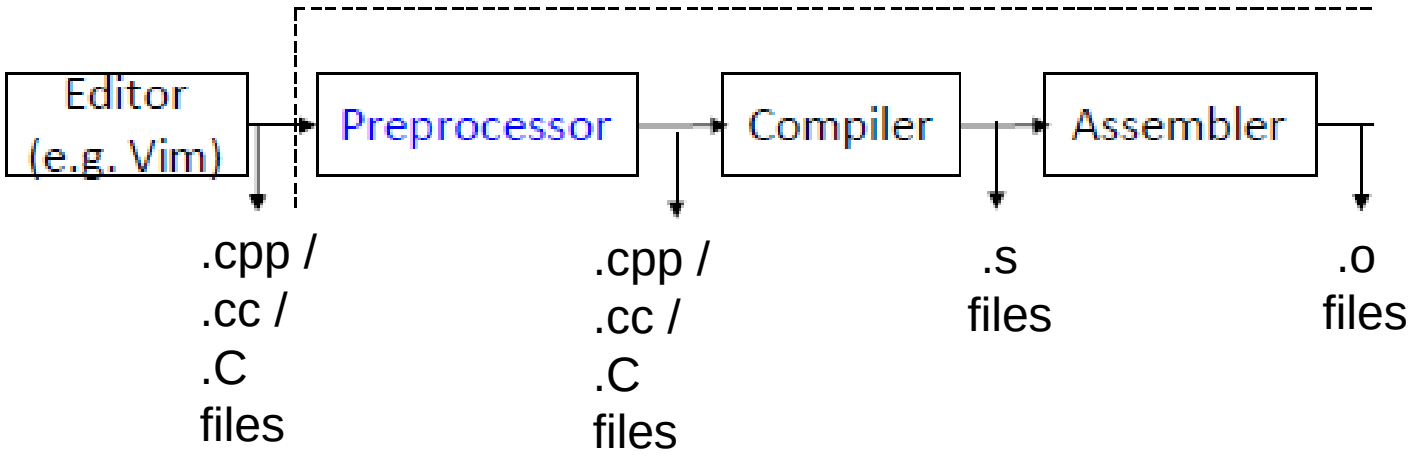
Creating a Program

- Translate your source code to assembly language



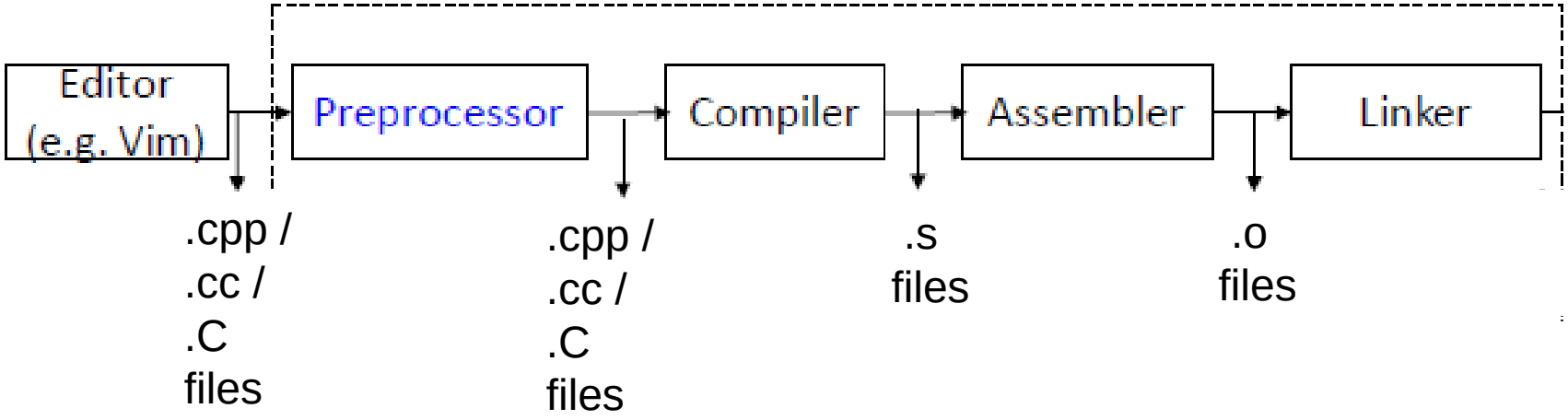
Creating a Program

- Translate your assembly code to machine code



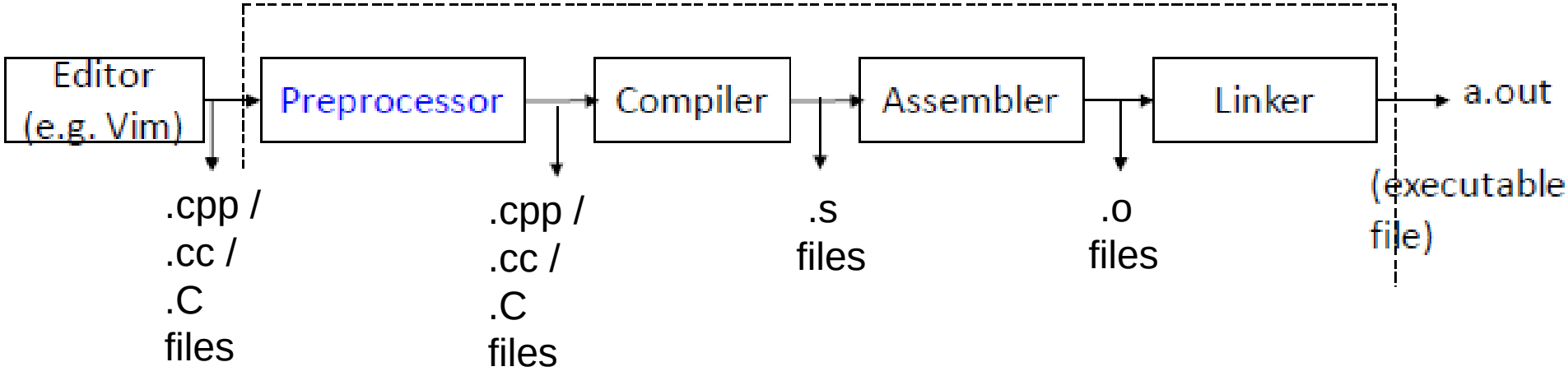
Creating a Program

- Get machine code that is part of libraries



Creating a Program

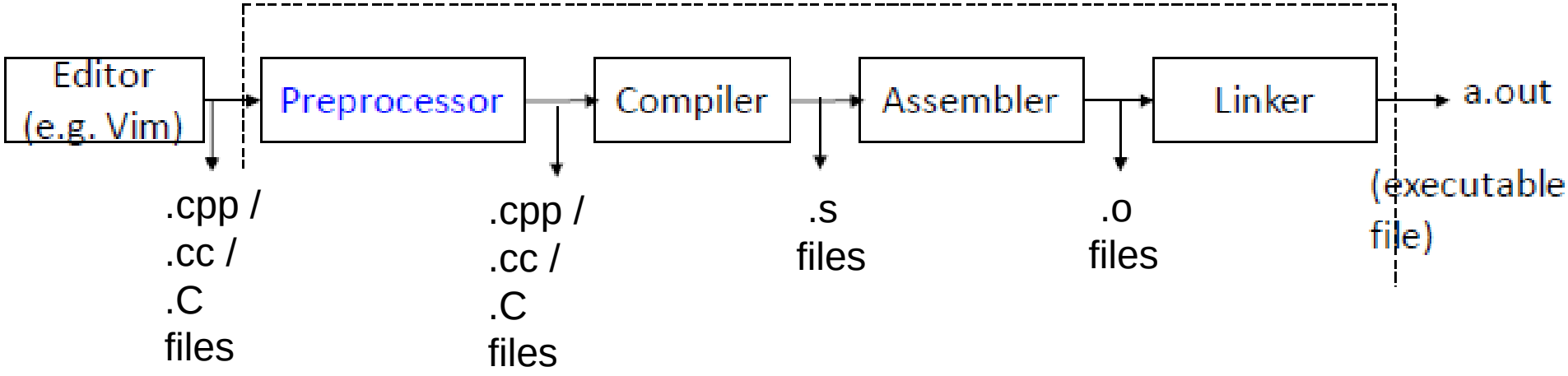
- Create executable



1. Either copy the corresponding machine code OR
2. Insert a ‘stub’ code to execute the machine code directly from within the library module

Creating a Program

- `g++ 4_8_1.cpp -lm`



  - g++ is a command to translate your source code (by invoking a collection of tools)
    - Above command produces `a.out` from `.cpp` file
  - `-l` option tells the linker to 'link' the math library

Creating a Program

- `g++:` other options
  - `-Wall` - Show all warnings
  - `-omyexe` - create the output machine code in a file called `myexe`
  - `-g` - Add debug symbols to enable debugging
  - `-c` - Just compile the file (don't link) i.e. produce a `.o` file
  - `-I/home/mydir` -Include directory called `/home/mydir`
  - -O1, -O2, -O3 – request to optimize code according to various levels

    Always check for program correctness when using optimizations

Creating a Program

- The steps just discussed are ‘compiled’ way of creating a program. E.g. C++
- Interpreted way: alternative scheme where source code is ‘interpreted’ / translated to machine code piece by piece e.g. MATLAB
- Pros and Cons.
  - Compiled code runs faster, takes longer to develop
  - Interpreted code runs normally slower, often faster to develop

Creating a Program

- For different parts of the program different strategies may be applicable.
  - Mix of compilation and interpreted – interoperability
- In the context of scientific software, the following are of concern:
  - Computational efficiency
  - Cost of development cycle and maintainability
  - Availability of high-performant tools / utilities
  - Support for user-defined data types

Creating a Program

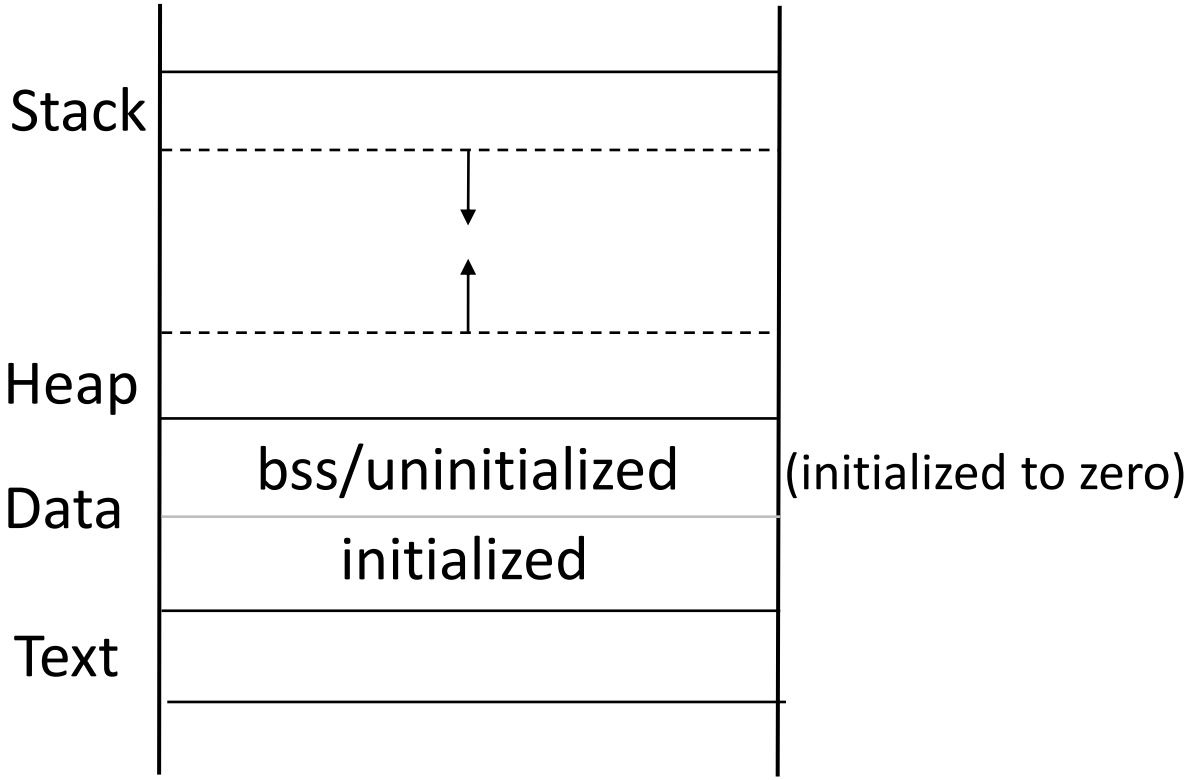
- `a.out` is a pattern of 0s and 1s laid out in memory – sequence of machine instructions
- How is a program laid out in memory?
  - Helpful to debug
  - Helpful to create robust software
  - Helpful to customize program for embedded systems

Program Layout in Memory

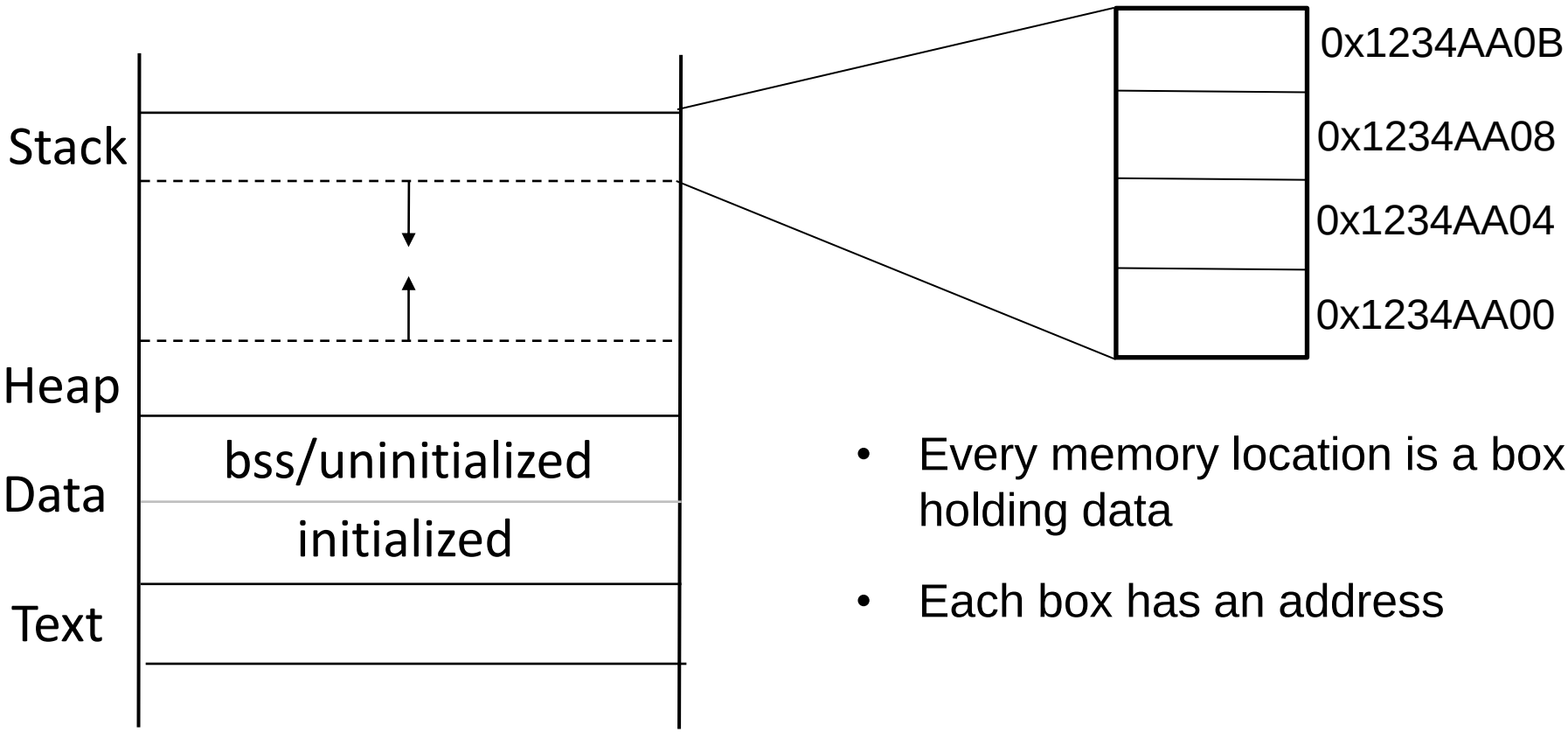
- A program's memory space is divided into four segments:

1. Text
   - source code of the program
2. Data
   - Broken into uninitialized and initialized segments; contains space for global and static variables. E.g. `int x = 7; int y;`
3. Heap
   - Memory allocated using malloc/calloc/realloc/new
4. Stack
   - Function arguments, return values, local variables, special registers.

Program Layout in Memory



Program Layout in Memory

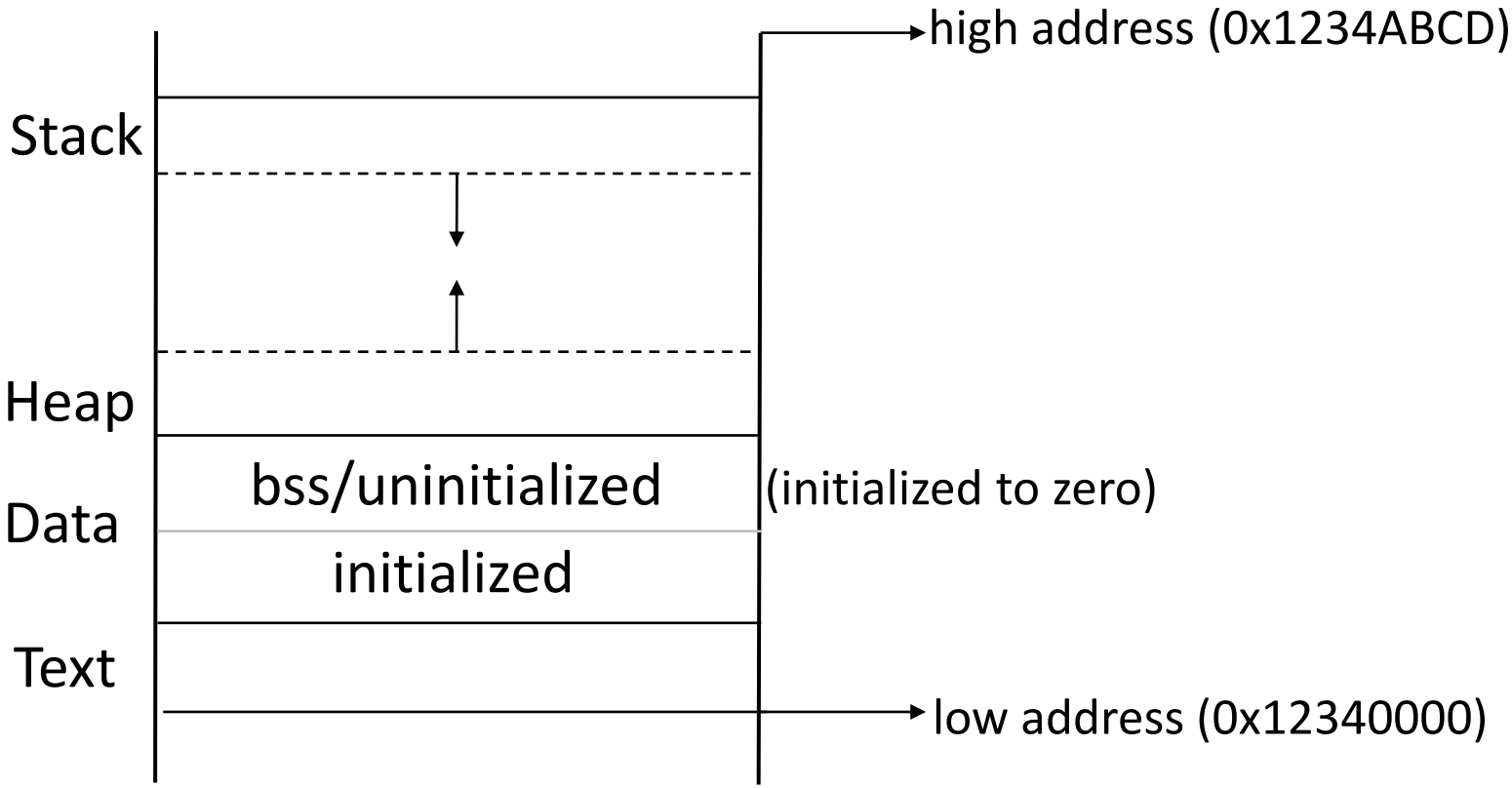


- Every memory location is a box holding data
- Each box has an address

Addresses

- Computer programs think and live in terms of memory locations
- Addresses in computer programs are just numbers identifying memory locations
- A program navigates by visiting one address after another

Program Layout in Memory



Addresses

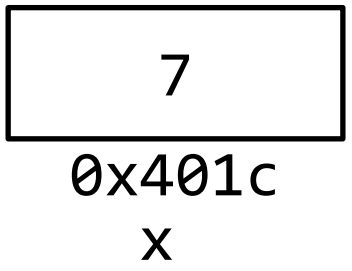
- Humans are not good at remembering numerical addresses.

what are the GPS coordinates (latitude and longitude) of your residence?

- We (humans) choose convenient ways to identify addresses so that we can give directions to a program. E.g. Variables

Handles to Addresses

- Variables
  - Its just a handle to an address / program memory location
- `int x = 7;`



- Read x => Read the content at address `0x401C`
- Write x=> Write at address `0x401C`

```
int x;
```

1. *What is the set of values this variable can take on in C?*

   -2^{31} to $(2^{31} - 1)$

2. *How much space does this variable take up?*

   32 bits

3. *How should operations on this variable be handled?*

   integer division is different from floating point divisions

```
3 / 2 = 1  //integer division

3.0 / 2.0 = 1.5  //floating-point division
```

C++ standard types

- Integer types: `char, short int, int, long int, long long int, bool`
- Float: `float, double, long double`
- `Pointers`: handle to addresses
- `References`: safer than pointers but less powerful
- `void`: nothing

C++ standard types

- Modifiers
  - `short, long, signed, unsigned.`
- Compound types
  - pointers, structs, enums, arrays, etc.

C++ standard types – storage space

Data type	Number of bytes
`char`	1
`short int`	2
`int / long int`	4
`long long int`	8
`float`	4
`double`	8
`long double`	12

- Use sizeof() operator to check the size of a type
  - e.g. `sizeof(int)`

Data types - quirks

- if no type is given compiler automatically converts it to `int` data type.
  - `signed x;`
- `long` is the only modifier allowed with `double`
  - `long double y;`
- `signed` is the default modifier for `char` and `int`
- Can't use any modifiers with `float`