Getting Started in C

- Lots will be familiar:
  - Declaring variables (mostly)
  - Loops
  - Conditionals
  - Functions
  - Including libraries
- But there are big differences (some)
  - Pointers
  - Memory management
  - Strings (or lack thereof)
  - No objects (structs only)
    - No polymorphism/inheritance/etc.
  - Print syntax is different
  - Compiler directives
  - Function prototypes

somecode.h

```
int getMax (int, int);
```

somecode.c

```
#include <stdio.h>
#include "somecode.h"
#define A 5
#define B 10
int getMax(int a, int b)
{
      if(a > b)
         return a;
      else
         return b;
}
int main(){
     printf("%d\n", getMax(A, B));
}
```

Common Practices Seen in C Source

- Sequence Operator **,**

  expr1*,*expr2

- Evaluates *expr1* and then *expr2* evaluates/returns to *expr2*

```
for (i = 0, j = 0; i < 10; i++, j++)
    …
```

- Assignment inside conditional test (this is very common!)

```
if ((i = SomeFunction()) != 0)
    statement1;
else
    statement2;
```

assignment returns the value that is placed into the variable to the left of the = sign, then the test is made

What does this code print when run as ./a.out 2?

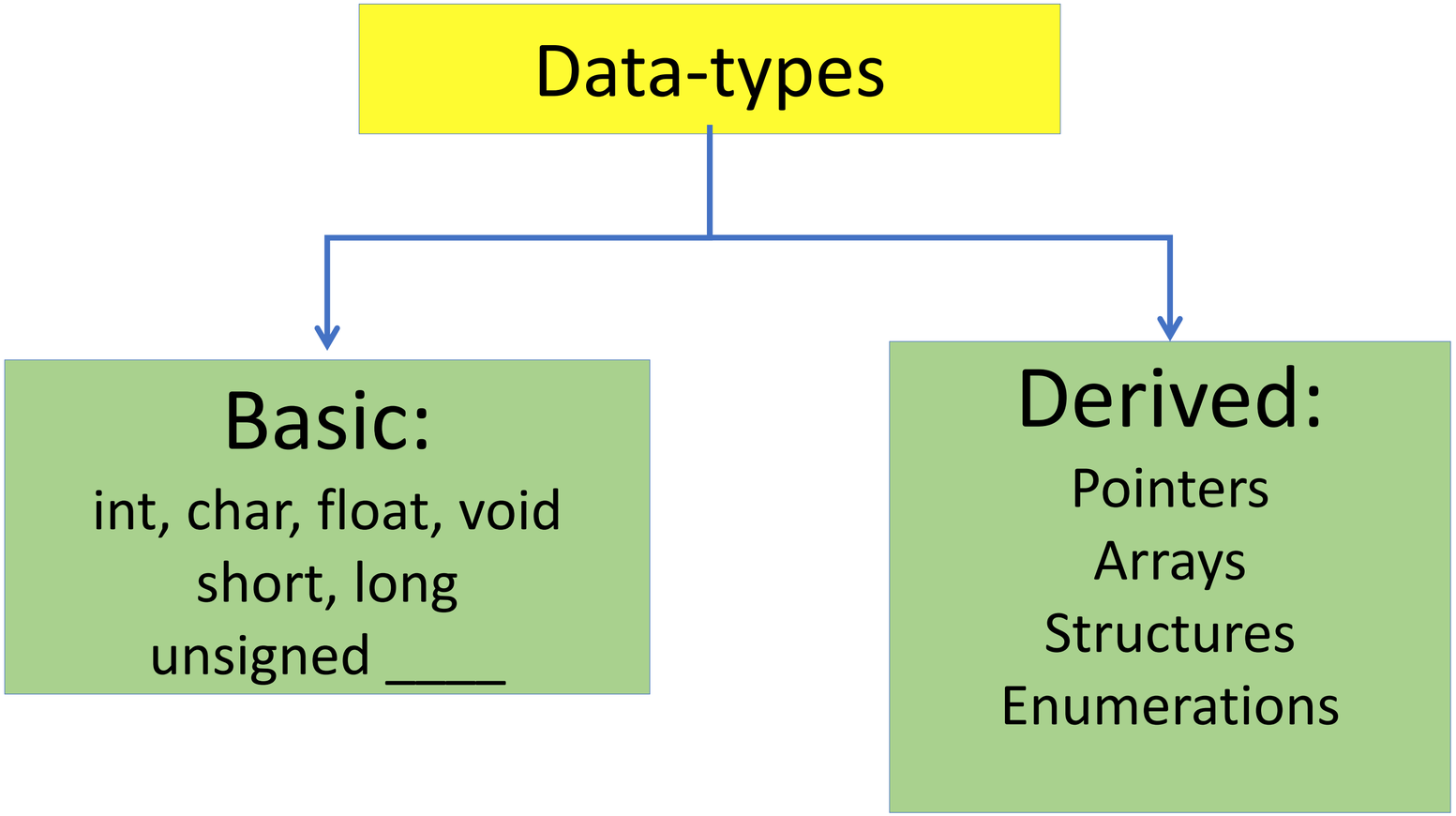
```
#include <stdio.h>
#include <stdlib.h>
int someFunction(int x){
   if (x = 4){
       x++;
   }
   return (x);
}

int main(int argc, char *argv[]){
  int someNum = atoi(argv[1]);
  printf("%d\n", someFunction(someNum));
}
```

A. 2
B. 3
C. 5
D. Won't compile
E. none of these

Data objects in C

Old IBM Disk Drive with visible platters

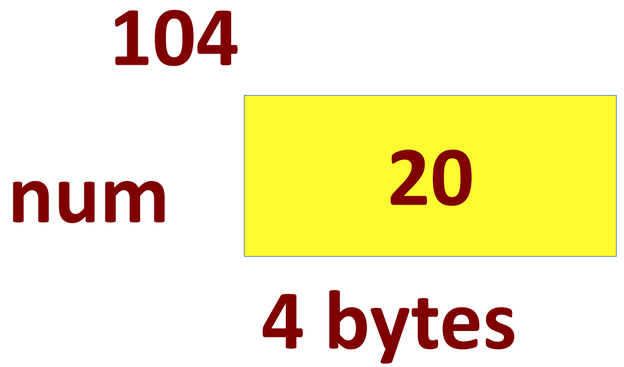
How we manipulate variables depends on *data-type*



Basic data object in memory

A region in memory that contains a value and is associated with name/identifier

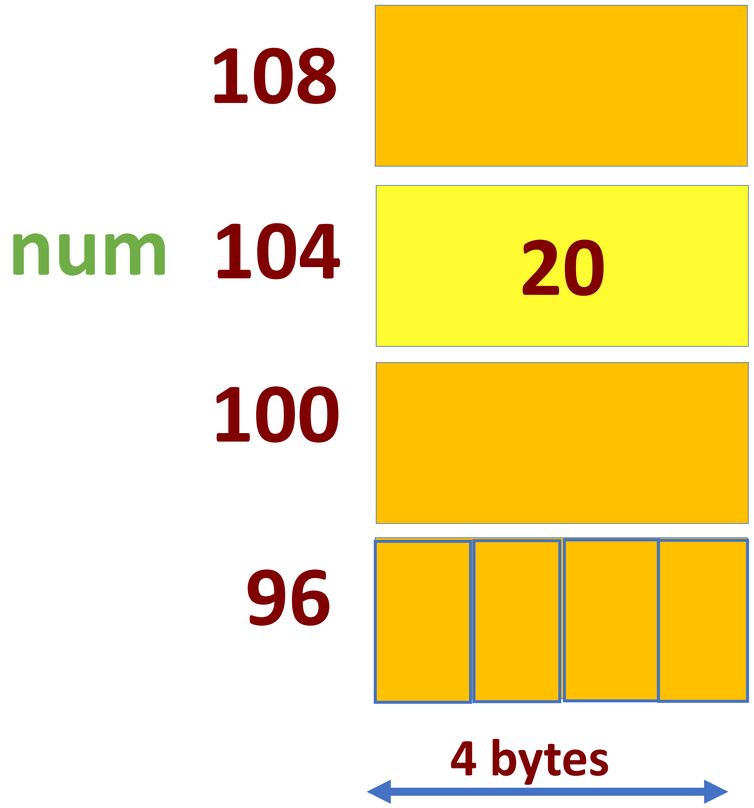
```
int
main(int argc, char**argv){

    int num;
    num = 20;
}
```



Basic data object in memory

A region in memory that contains a value and is associated with name/identifier

```
int
main(int argc, char**argv){
    int num;
    num = 20;
}
```



Attributes of a Data-object/variable:

- Name/identifier
- Value
- Address: Location in memory
- Size
- *data-type*
- Lifetime
- Scope

Definition vs Declaration

Definition

what is <it> and create an instance.

- function: create storage for it and corresponding code
- variable: create storage and put value there (optional)
- **only define once!**

```
int a;
short sum(short a, short b){
  return (a + b);
}
```

Declaration

describe <it>

- e.g function prototype
- variable named but defined elsewhere), containers for data (called structs)

```
extern int a;
short sum(short, short);
```

Declaration & Definition

What are these statement(s)?

```
extern int func(int, int);   // I
```

```
int func2(int a, int b) {    // II
    return a-b;              // II
}                            // II
```

Example definitions (some with initialization)

```
char c='a';              // 1 byte
short s;                 // 2 bytes
int a;                   // usually 4 bytes  - signed
unsigned int a=0;        // usually 4 bytes
float f;                 // 4 bytes use sizeof(float)
double d;                // 8 bytes use sizeof(double)
long double d;           // quad fl. pt. usually 16 bytes)
```

Header Files

- Include Header files (.h) that contain function declarations - the function interface
  - Function declaration (return type, argument types)
- Some other .c files contain the actual code (definition)
- Include files (.h) contain variables referenced here but defined elsewhere (later)

somecode.h

```
int getMax (int, int);
extern int someGlobalVar;
```

somecode.c

```
#include <stdio.h>
#include "somecode.h"
#define A 5
#define B 10
int someGlobalVar = 10;
int getMax(int a, int b)
{
      if(a > b)
         return a;
      else
         return b;
}
int main(){
    printf("%d\n", getMax(A, B));
}
```

function definition

Which of the following are NOT appropriate for a header file?

```
int a = 10;                    // I
int b;                         // II
extern int c;                  // III
char rotateMe(char c);         // IV
```

A. I.
B. II.
C. I. && II.
D. III. && IV.
E. IV.