# CS 33

## Introduction to C
### Part 4


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Number Representation

- Hindu-Arabic numerals
  - developed by Hindus starting in 5th century
    - » positional notation
    - » symbol for 0
  - adopted and modified somewhat later by Arabs
    - » known by them as “Rakam Al-Hind” (Hindu numeral system)
  - 1999 rather than MCMXCIX
    - » (try doing long division with Roman numerals!)

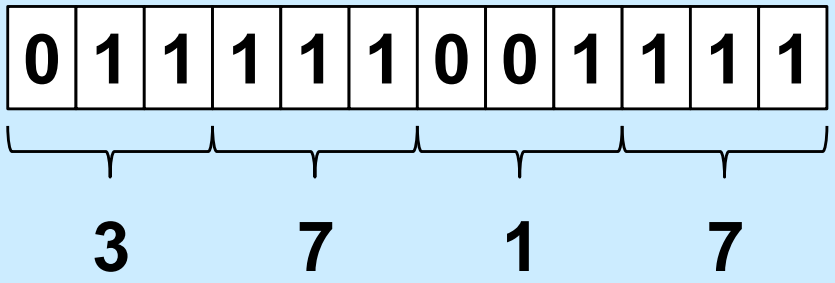

Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Which Base?

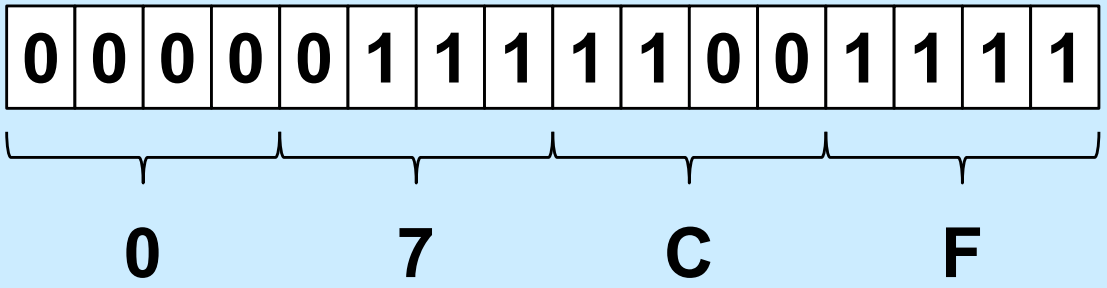
- 1999
  - base 10
    - » $9 \cdot 10^0 + 9 \cdot 10^1 + 9 \cdot 10^2 + 1 \cdot 10^3$
  - base 2
    - » 11111001111
      - $1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 + 0 \cdot 2^5 + 1 \cdot 2^6 + 1 \cdot 2^7 + 1 \cdot 2^8 + 1 \cdot 2^9 + 1 \cdot 2^{10}$
  - base 8
    - » 3717
      - $7 \cdot 8^0 + 1 \cdot 8^1 + 7 \cdot 8^2 + 3 \cdot 8^3$
    - » why are we interested?
  - base 16
    - » 7CF
      - $15 \cdot 16^0 + 12 \cdot 16^1 + 7 \cdot 16^2$
    - » why are we interested?


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Words …




**12-bit computer word**




**16-bit computer word**


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Base Conversion Algorithm

```
void baseX(unsigned int num, unsigned int base) {
   char digits[] = {'0', '1', '2', '3', '4', '5', '6', … };
   char buf[8*sizeof(unsigned int)+1];
   int i;

   for (i = sizeof(buf) - 2; i >= 0; i--) {
      buf[i] = digits[num%base];
      num /= base;
      if (num == 0)
         break;
   }

   buf[sizeof(buf) - 1] = '\0';
   printf("%s\n", &buf[i]);
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Or ...

```
$ bc
obase=16
1999
7CF
$
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Quiz 1

- **What's the decimal (base 10) equivalent of $25_{16}$?**
  - **a) 19**
  - **b) 35**
  - **c) 37**
  - **d) 38**


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Encoding Byte Values

- **Byte = 8 bits**
  - binary $00000000_2$ to $11111111_2$
  - octal $0_8$ to $377_8$
    - » write $377_8$ in C as
      - 0377
  - decimal: $0_{10}$ to $255_{10}$
  - hexadecimal $00_{16}$ to $FF_{16}$
    - » base 16 number representation
    - » use characters '0' to '9' and 'A' to 'F'
    - » write $FA1D37B_{16}$ in C as
      - 0xFA1D37B
      - 0xfa1d37b

| Hex | Decimal | Octal | Binary |
|---|---|---|---|
| 0 | 0 | 0 | 0000 |
| 1 | 1 | 1 | 0001 |
| 2 | 2 | 2 | 0010 |
| 3 | 3 | 3 | 0011 |
| 4 | 4 | 4 | 0100 |
| 5 | 5 | 5 | 0101 |
| 6 | 6 | 6 | 0110 |
| 7 | 7 | 7 | 0111 |
| 8 | 8 | 10 | 1000 |
| 9 | 9 | 11 | 1001 |
| A | 10 | 12 | 1010 |
| B | 11 | 13 | 1011 |
| C | 12 | 14 | 1100 |
| D | 13 | 15 | 1101 |
| E | 14 | 16 | 1110 |
| F | 15 | 17 | 1111 |

# Unsigned 32-Bit Integers


$$\text{value} = \sum_{i=0}^{31} b_i \cdot 2^i$$

(we ignore negative integers for now)


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Storing and Viewing Ints

```
int main() {
    unsigned int n = 57;
    printf("binary: %b, decimal: %u, "
           "hex: %x\n", n, n, n);
    return 0;
}
```

```
$ ./a.out
binary: 111001, decimal: 57, hex: 39
$
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Boolean Algebra

- **Developed by George Boole in 19th Century**
  - **algebraic representation of logic**
    - » **encode "true" as 1 and "false" as 0**

## And

- A&B = 1 when both A=1 and B=1

| & | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

## Or

- A|B = 1 when either A=1 or B=1

| \| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

## Not

- ~A = 1 when A=0

| ~ | |
|---|---|
| 0 | 1 |
| 1 | 0 |

## Exclusive-Or (Xor)

- A^B = 1 when either A=1 or B=1, but not both

| ^ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

# General Boolean Algebras

- Operate on bit vectors
  - operations applied bitwise

```
  01101001     01101001     01101001
& 01010101   | 01010101   ^ 01010101   ~ 01010101
  --------     --------     --------     --------
  01000001     01111101     00111100     10101010
```

- All of the properties of boolean algebra apply

# Example: Representing & Manipulating Sets

- **Representation**
  - width-w bit vector represents subsets of {0, …, w–1}
  - $a_j = 1$ iff $j \in A$

    01101001 { 0, 3, 5, 6 }
    *76543210*

    01010101 { 0, 2, 4, 6 }
    *76543210*

- **Operations**

| | | | |
|---|---|---|---|
| & | intersection | 01000001 | { 0, 6 } |
| \| | union | 01111101 | { 0, 2, 3, 4, 5, 6 } |
| ^ | symmetric difference | 00111100 | { 2, 3, 4, 5 } |
| ~ | complement | 10101010 | { 1, 3, 5, 7 } |

# Bit-Level Operations in C

- Operations &, |, ~, ^ available in C
  - apply to any "integral" data type
    - » long, int, short, char
  - view arguments as bit vectors
  - arguments applied bit-wise
- Examples (char datatype)

~0x41 → 0xBE

$\sim 01000001_2 \rightarrow 10111110_2$

~0x00 → 0xFF

$\sim 00000000_2 \rightarrow 11111111_2$

0x69 & 0x55 → 0x41

$01101001_2 \;\&\; 01010101_2 \rightarrow 01000001_2$

0x69 | 0x55 → 0x7D

$01101001_2 \mid 01010101_2 \rightarrow 01111101_2$

# Contrast: Logic Operations in C

- **Contrast to Logical Operators**
  - `&&, ||, !`
    - » view 0 as “false”
    - » anything nonzero as “true”
    - » always return 0 or 1
    - » early termination/short-circuited execution
- **Examples (char datatype)**

```
!0x41 → 0x00
!0x00 → 0x01
!!0x41 → 0x01

0x69 && 0x55 → 0x01
0x69 || 0x55 → 0x01
p && complicated_function(x)
```

# Contrast: Logic Operations in C

- **Contrast to Logical Operators**
  - **&&, ||, !**
    - **» vie… “false”**

**Watch out for && vs. & (and || vs. |)… One of the more common oopsies in C programming**

- **L…**

```
!0x41 → 0x00
!0x00 → 0x01
!!0x41 → 0x01

0x69 && 0x55 → 0x01
0x69 || 0x55 → 0x01
p && complicated_function(x)
```

# Quiz 2

- **Which of the following would determine whether the next-to-the-rightmost bit of Y (declared as a char) is 1? (I.e., the expression evaluates to true if and only if that bit of Y is 1.)**
  a) **Y & 0x02**
  b) **!((~Y) & 0x02)**
  c) **none of the above**
  d) **both a and b**


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Shift Operations

- **Left Shift:** `x << y`
  - shift bit-vector x left y positions
    - throw away extra bits on left
    - » fill with 0's on right
- **Right Shift:** `x >> y`
  - shift bit-vector x right y positions
    - » throw away extra bits on right
  - logical shift
    - » fill with 0's on left
  - arithmetic shift
    - » replicate most significant bit on left
- **Undefined Behavior**
  - shift amount < 0 or ≥ word size

| Argument x | 01100010 |
|---|---|
| << 3 | 00010*000* |
| Log. >> 2 | *00*011000 |
| Arith. >> 2 | *00*011000 |

| Argument x | 10100010 |
|---|---|
| << 3 | 00010*000* |
| Log. >> 2 | *00*101000 |
| Arith. >> 2 | *11*101000 |

# Digression

- **Pre-increment**
  - **++*b* means add one to *b;* the result of the expression is this new value of *b***
- **Post-increment**
  - ***b*++ means the value of the expression is the current value of *b*, then add one to *b***
- **Example**

```
int b=1;
printf("%d\n", (++b)*b);
```

**output:**

```
4
```

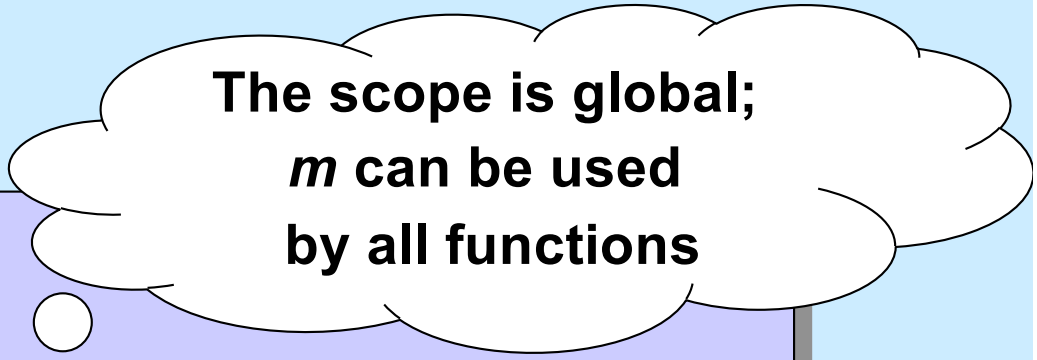
```
int b=1;
printf("%d\n", (b++)*b);
```

**output:**

```
2
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Global Variables




```
#define NUM_ROWS 3
#define NUM_COLS 4
int m[NUM_ROWS][NUM_COLS];

int main() {
    int row, col;
    for(row=0; row<NUM_ROWS; row++)
      for(col=0; col<NUM_COLS; col++)
          m[row][col] = row*NUM_COLS+col;
    return 0;
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Global Variables

```
#define NUM_ROWS 3
#define NUM_COLS 4
int m[NUM_ROWS][NUM_COLS];

int main() {
    int row, col;
    printf("%u\n", m);
    printf("%u\n", &row);
    return 0;
}
```

```
$ ./a.out
8384
3221224352
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Global Variables are Initialized!

```
#define NUM_ROWS 3
#define NUM_COLS 4
int m[NUM_ROWS][NUM_COLS];

int main() {
    printf("%d\n", m[0][0]);
    return 0;
}
```

```
$ ./a.out
0
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Scope

```
int a;   // global variable

int main() {
   int a;   // local variable
   a = 0;
   proc();
   printf("a = %d\n", a); // what's printed?
   return 0;
}

int proc() {
   a = 1;
   return a;
}
```

```
$ ./a.out
0
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Scope (continued)

```
int a;   // global variable

int main() {
   a = 2;
   proc(1);
   return 0;
}

int proc(int a) {
   printf("a = %d\n", a); // what's printed?
   return a;
}
```

```
$ ./a.out
1
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Scope (still continued)

```
int a;   // global variable

int main() {
   a = 2;
   proc(1);
   return 0;
}

int proc(int a) {
   int a;
   printf("a = %d\n", a); // what's printed?
   return a;
}
```

```
$ gcc prog.c
prog.c:12:8: error: redefinition of 'a'
   int a;
       ^
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Scope (more ...)

```
int a;   // global variable

int proc() {
   {
      // the brackets define a new scope
      int a;
      a = 6;
   }
   printf("a = %d\n", a); // what's printed?
   return 0;
}
```

```
$ ./a.out
0
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Quiz 3

```
int a;

int proc(int b) {
    {int b=6;}
    a = b;
    return a+2;
}

int main() {
    {int a = proc(4);}
    printf("a = %d\n", a);
    return 0;
}
```

- **What's printed?**
  a) **0**
  b) **4**
  c) **6**
  d) **8**
  e) **nothing; there's a syntax error**


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Scope and For Loops (1)

```
int A[100];
for (int i=0; i<100; i++) {
  // i is defined in this scope
  A[i] = i;
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Scope and For Loops (2)

```
int A[100];
initializeA(A);
for (int i=0; i<100; i++) {
  // i is defined in this scope
  if (A[i] < 0)
    break;
}
if (i != 100)
  printf("A[%d] is negative\n", i);
```

**syntax error: reference to *i* is out of scope.**


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Lifetime

```
int count;

int main() {
    func();
    ...
    func(); // what's printed by func?
    return 0;
}

int func() {
    int a;
    if (count == 0) a = 1;
    count = count + 1;
    printf("%d\n", a);
    return 0;
}
```

```
% ./a.out
1
-38762173
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Lifetime (continued)

```
int main() {
    func(1); // what's printed by func?
    return 0;
}
int a;
int func(int x) {
    if (x == 1) {
        a = 1;
        func(2);
        printf("%d\n", a);
    } else
        a = 2;
    return 0;
}
```

```
% ./a.out
2
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Lifetime (still continued)

```
int main() {
    func(1); // what's printed by func?
    return 0;
}

int func(int x) {
    int a;
    if (x == 1) {
        a = 1;
        func(2);
        printf("a = %d\n", a);
    } else
        a = 2;
    return 0;
}
```

```
% ./a.out
1
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Lifetime (more ...)

```
int main() {
    int *a;
    a = func();
    printf("%d\n", *a); // what's printed?
    return 0;
}

int *func() {
    int x;
    x = 1;
    return &x;
}
```

```
% ./a.out
23095689
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Lifetime (and still more ...)

```
int main() {
    int *a;
    a = func(1);
    printf("%d\n", *a); // what's printed?
    return 0;
}

int *func(int x) {
    return &x;
}
```

```
% ./a.out
98378932
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

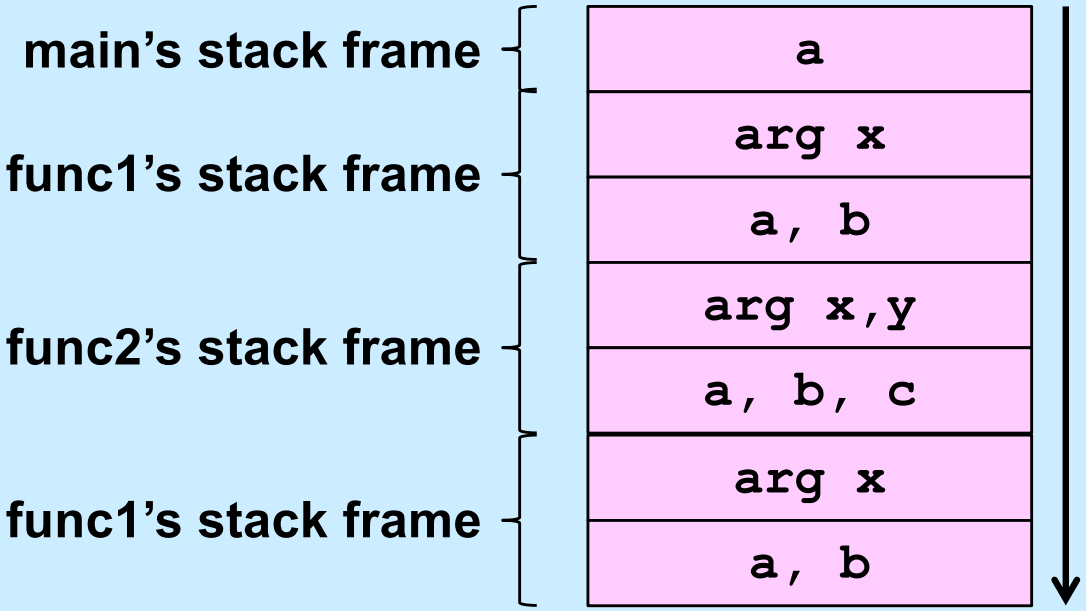
# Rules

- Global variables exist for the duration of program's lifetime
- Local variables and arguments exist for the duration of the execution of the function
  - from call to return
  - each execution of a function results in a new instance of its arguments and local variables


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Implementation: Stacks

```
int main() {
    int a;
    func1(0);
    ...
}
int func1(int x) {
    int a,b;
    if (x==0) func2(a,2);
    ...
}
int func2(int x, int y) {
    int a,b,c;
    func1(1);
    ...
}
```

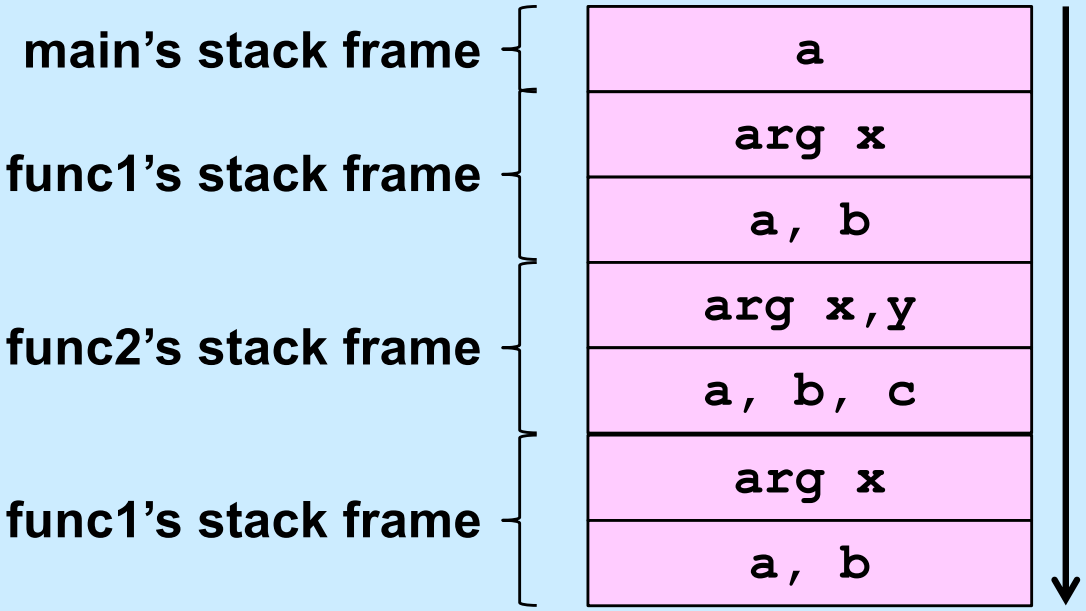




Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Implementation: Stacks

```
int main() {
    int a;
    func1(0);
    ...
}
int func1(int x) {
    int a,b;
    if (x==0) func2(a,2);
    ...
}
int func2(int x, int y) {
    int a,b,c;
    func1(1);
    ...
}
```





Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Quiz 4

```
void func(int a) {
    int b=2;
    if (a == 1) {
        func(2);
        printf("%d\n", b);
    } else {
        b = a*(b++)*b;
    }
}
int main() {
    func(1);
    return 0;
}
```

- **What's printed?**
  - **a) 0**
  - **b) 1**
  - **c) 2**
  - **d) 4**


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Static Local Variables

```
int *sub1() {
  int var = 1;
  …
  return &var;
  /* amazingly illegal */
}
```

```
int *sub2() {
  static int var = 1;
  …
  return &var;
  /* (amazingly) legal */
}
```

- **Scope**
  - **like local variables**
- **Lifetime**
  - **like global variables**
- **Initialized just once**
  - **when program begins**
  - **implicit initialization to 0**


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Quiz 5

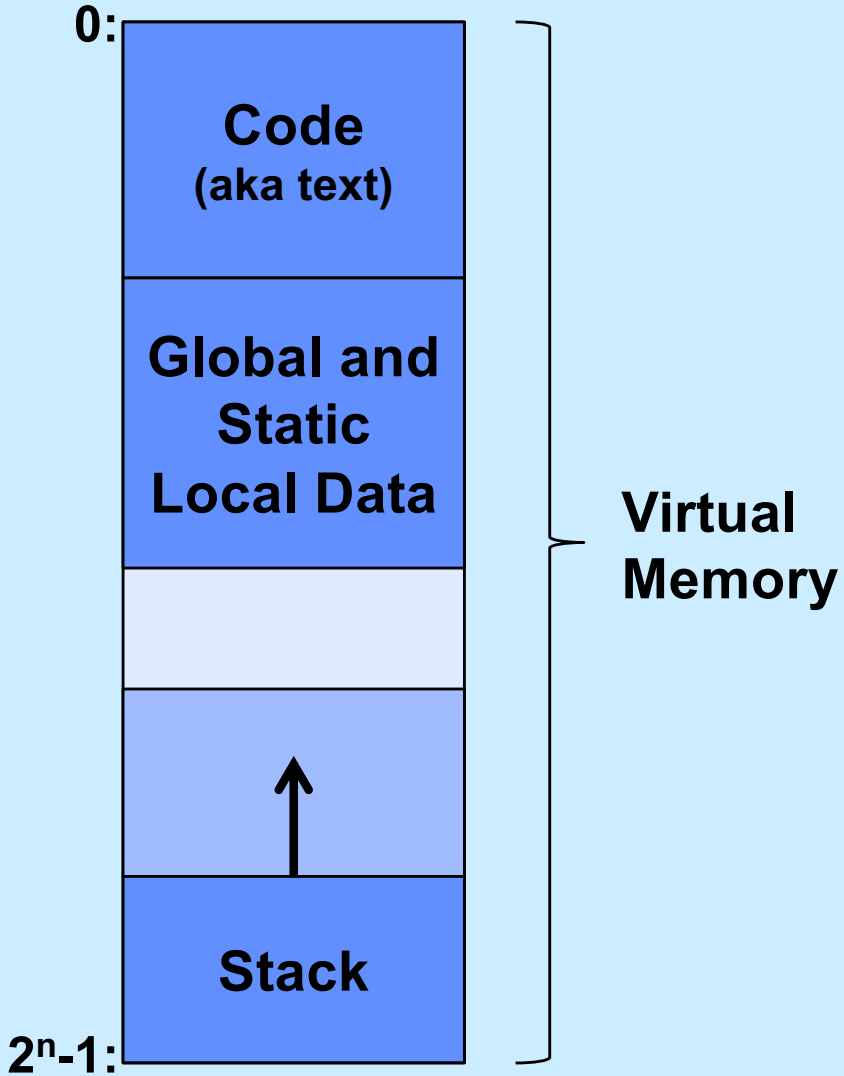
```
int sub() {
  static int svar = 2;
  int lvar = 1;
  svar += lvar;
  lvar++;
  return svar;
}

int main() {
  sub();
  printf("%d\n", sub());
  return 0;
}
```

**What is printed?**

a) 2
b) 3
c) 4
d) 5


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Digression: Where Stuff Is
## (Roughly)





Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# scanf: Reading Data

```
int main() {
    int i, j;
    scanf("%d %d", &i, &j);
    printf("%d, %d", i, j);
}
```

```
$ ./a.out
   3          12
3, 12
```

**Two parts**

- **formatting instructions**
  - **whitespace in format string matches any amount of white space in input**
    - » **whitespace is space, tab, newline (‘\n’)**
- **arguments: must be addresses**
  - **why?**


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# #define (again)

```
#define CtoF(cel) (9.0*cel)/5.0 + 32.0
```

**Simple textual substitution:**

```
float tempc = 20.0;
float tempf = CtoF(tempc);
// same as tempf = (9.0*tempc)/5.0 + 32.0;
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Careful ...

```
#define CtoF(cel) (9.0*cel)/5.0 + 32.0

float tempc = 20.0;
float tempf = CtoF(tempc+10);
// same as tempf = (9.0*tempc+10)/5.0 + 32.0;
```

```
#define CtoF(cel) (9.0*(cel))/5.0 + 32.0

float tempc = 20.0;
float tempf = CtoF(tempc+10);
// same as tempf = (9.0*(tempc+10))/5.0 + 32.0;
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.

# Conditional Compilation

```
#ifdef DEBUG
  #define DEBUG_PRINT(a1, a2) printf(a1,a2)
#else
  #define DEBUG_PRINT(a1, a2)
#endif
```

```
int buggy_func(int x) {
    DEBUG_PRINT("x = %d\n", x);
      // printed only if DEBUG is defined
    ...
}
```


Copyright © 2023 Thomas W. Doeppner. All rights reserved.