Introduction to C, C++, and Unix/Linux

CS 60

Lecture 3: Data types and variables

Today

→ C data types and variables

- Reading for Next Time: KR Chapters 1-3 & 7.1-7.4

Notes

- Questions?

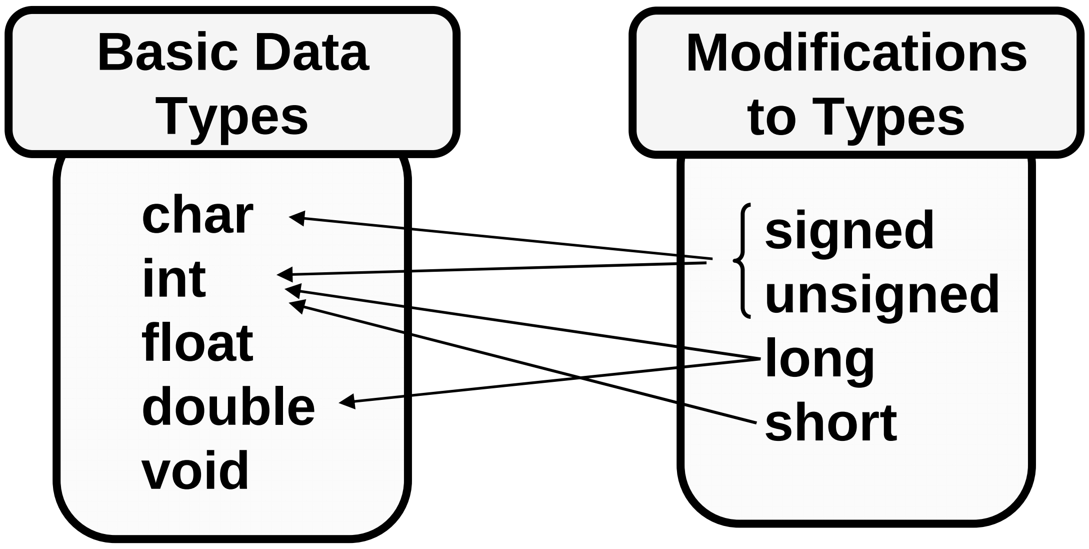
Lexical Elements in C

- **Keywords**
  - Reserved words that may not be used for anything else
- **Identifiers**
  - Variable names, function names...
- **Constants**
  - E.g., the number 5
- **String constants**
  - E.g. “Hello, world\n”
- **Operators**
  - E.g., +, -, =, ++
- **Punctuators**
  - E.g., {} ( ) ; ,

These are the basic tokens that the compiler cares about

Data types

- ANSI C has five "atomic" data types, and several modifications to the atomic types



(unsigned, signed) char
(unsigned, signed) (short, long) int
float
(long) double
void

22 basic data types:

char
unsigned char
signed char
short int
unsigned short int
signed short int
short

unsigned short
signed short
int
unsigned int
signed int
long int
unsigned long int

signed long int
long
unsigned long
signed long
float
double
long double
void

How big is a…?

Returns the number of <u>bytes</u> used to store the argument type

- char (signed, unsigned)
- int (signed, unsigned) (long, short)
- float
- double (long)
- void

Use sizeof() function:

```
nbytes = sizeof(x);
nbytes = sizeof(long int);
nbytes = sizeof(double);
```

sizeof() argument can be variable name or type

```
printf("%d", sizeof(char));           1
printf("%d", sizeof(short int));      2
printf("%d", sizeof(int));            4   ←
printf("%d", sizeof(long int));       4
printf("%d", sizeof(float));          4
printf("%d", sizeof(double));         8   ←
printf("%d", sizeof(long double));   12
printf("%d", sizeof(void));           1
void *ptr;
printf("%d", sizeof(ptr));            4
```

Range of values?

char	-2^7 to (2^7-1)
unsigned char	0 to (2^8-1)
short int	-2^{15} to $(2^{15}-1)$
unsigned short int	0 to $(2^{16}-1)$
int	-2^{31} to $(2^{31}-1)$
unsigned int	0 to $(2^{32}-1)$
address pointer	0 to $(2^{32}-1)$
	(4 GB)

Range of values?

char	-128…127
unsigned char	0…255
short int	-32,768…32,767
unsigned short int	0 to 65,535
int	-2,147,483,648…2,147,483,647
unsigned int	0…4,294,967,296
address pointer	0…4,294,967,296

`if (-1) printf(“YES”);`

C has no boolean type

- Logical and relational operators like **`==`**, **`||`**, **`!`**, **`!=`**, etc. return an integer
  - 1 if TRUE
  - 0 if FALSE
- When checking a boolean relation (e.g., if/then)
  - 0 means FALSE
  - Non-zero means TRUE
    - ♦ -1 means TRUE!

So it’s a little strange that if **`main()`** completes successfully it returns 0!

Type conversion

- When an operator has operands of different types, or a function gets a type different from that specified, type conversion occurs (if possible)

```
int x=1;                double x=1;
double y = x;           int y = x;
```

```
int x=365;
char c = x;
```

Casting

```
x = (int) 3.2;
y = (double) x;
c = (char) x
```

- To avoid compiler errors and/or warnings, and to show that you *mean* to do so, you can explicitly force one type into another by *casting*

```
int x=1;                    double x=1;
double y = (double) x;      int y = (int) x;
```

```
int x=365;
char c = (char) x;
```

Number constants

`1` means **`(int)1`**

`1L` means **`(long)1`**

`3U` means **`(unsigned int)3`**

`3F` means **`(float)3`**

- An integer constant by itself is assume to be of type **`int`**
  - Or, if necessary, **`long`** or **`unsigned long`**
- If you want otherwise, append with:
  - **`L`** (or **`l`**) to make it **`long`**
  - **`U`** (or **`u`**) to make it **`unsigned`**
  - **`F`** (or **`f`**) to make it **`float`**

`1.2` means **`(double)1.2`**

`1.2F` means **`(float)1.2`**

`1.2L` means **`(long double)1.2`**

- A real constant by itself is assume to be of type **`double`**
- If you want otherwise, append with:
  - **`F`** (or **`f`**) to make it **`float`**
  - **`L`** (or **`l`**) to make it **`long double`**
- Can use e notation: **`123.45e-2`** $\rightarrow$ **`1.2345`**

String constants

- A string constant (string literal) is a sequence of characters enclosed by double quotes
  - Stored as an <u>array</u> of type **`char`**
  - Last array element is a zero

`“Hello, world\n”`

`H`	`e`	`l`	`l`	`o`	`,`		`w`	`o`	`r`	`l`	`d`	`\n`	`\0`
72	101	108	108	111	44	32	119	111	114	108	100	10	0

String

- Examples:
  - **`“Hello, world”`** Length: 12 (plus a zero)
  - **`“Hello, world\n”`** Length: 13 (plus a zero)
  - **`“Hello, world\0”`** Length: 12 (plus two zeros)
  - **`“This ” “is a string”`** → **`This is a string`**
  - **`“He said, \”Holy cow!\””`**

    → **`He said, “Holy cow!”`**

const

```
const int x = 8933849;
```

- Defines **`x`** as a constant variable
- Changing **`x`** causes a compiler warning

- Often used in function definitions with pointers...

enum

- Enumerated type
  - Defines a range of related constants
  - By default, the first is set to 0, and subsequent entries are incremented by 1 (values are generated for you)

```
enum {FALSE, TRUE};
enum {FALSE=0, TRUE};
enum {FALSE=0, TRUE=1};
```

These are all the same

Can name the enum here

enum

$\boxed{x = 01100101_2 = 101}$

```
enum {cs=100, ece, mat, chem, mech};
enum {bit1=1, bit2=2, bit3=4, bit4=8,
      bit5=16, bit6=32, bit7=64,
      bit8=128};
enum animals {dog, cat, bird, rat};

int x = bit1 | bit3 | bit6 | bit7;
```

Bases

$\boxed{\text{0x13 = 023 = 19}}$

- Base 10 is the default
- Base 8 (octal): Lead with a zero
  - 010, 023, 055
- Base 16 (hex0): Lead with 0x
  - 0x24, 0xf3, 0xcd0a5

- C doesn’t do base 2!

Variables

Memory is allocated on the stack

Compiler is told to expect var

Values are assigned

Variables are *defined*, *declared*, and *assigned*

```
int x;
double x, y;
```

```
x = 100;
x = y = 3.14;
```

```
extern x;
int function(int x) { ... }
```

More on this later

Simultaneous definition and assignment

int x = 99;

How about:

    int x, y = 99;

No value to x

    int x=y=99;

Error

int y;
int x=y=99;

Sets both **x** and **y** to the value 99