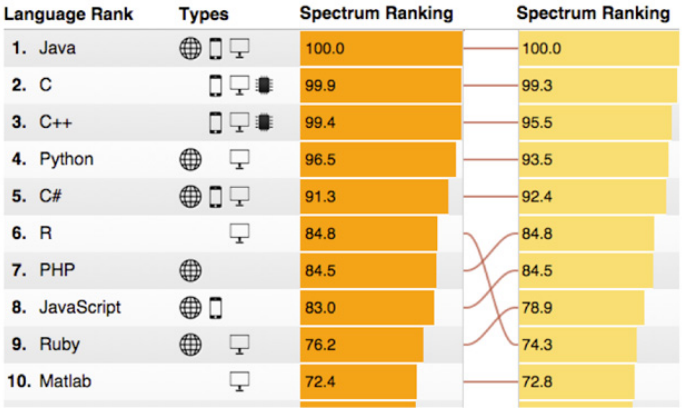
Introduction to C

EECS 2031

History

- C and Unix: widely used, powerful, and fast.
- Both started at AT&T Bell Labs.
- Unix was written in assembly, later changed to C.
- Many variants of Unix.

2015 Top 10 Programming Languages



- Source: http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages
- Ranking method: weighting and combining 12 metrics from 10 data sources (IEEE Xplore digital library, GitHub, CareerBuilder, etc.)

3

C vs. Java

- Java-like (actually Java has a C-like syntax), some differences
- No //, only /* */ multi-line and no nesting
- No garbage collection
  - Remember to free unused space yourself!
- No classes
- No exceptions (try … catch)
- No type “String”
  - string = array of characters ended by a null character ‘\0’

4

First C Program

```
#include <stdio.h>
main() {
   printf("hello, world \n");
}
```

- `#include <filename.h>` replaces the line by the actual file before compilation starts.
- `stdio.h`: defines most of the C file I/O functions

5

Compile and Run

```
> cc hello.c
> a.out
hello, world

> cc -o hello hello.c
> hello
hello, world
```

6

Special Characters

\n	New line
\t	Tab
\”	Double quote
\\	The \ character
\0	The null character
\’	Single quote

7

More Examples

- Many examples are given in the textbook, Chapter 1.
- To be discussed in class and in weekly labs.

8

Basic Input and Output

EECS 2031

9

Basic I/O

- Every program has a *standard input* and *output.*
- Usually, keyboard and monitor, respectively.
- Can use > and < for redirection
- getchar()
- putchar()
- printf()
- scanf()

Tidbit: How to pronounce “char” in “getchar”?

http://english.stackexchange.com/questions/60154/how-to-pronounce-the-programmers-abbreviation-char

10

getchar( ) (7.1)

- To read one character at a time from the *standard input* (the keyboard by default):

```
int getchar(void)
```

- returns the next input character each time it is called;
- returns EOF when it encounters end of file.
  - EOF input: Ctrl-d (Unix) or Ctrl-z (Windows).
  - EOF value defined in <stdio.h> is −1.

11

putchar(c) (7.1)

- Puts the character c on the *standard output* (the screen by default).

```
int putchar(int)
```

- returns the character written;
- returns EOF if an error occurs.

12

Example

```
#include <stdio.h>
#include <ctype.h>
main() /* convert input to lower case*/
{
 int c;
 c = getchar();
 while ( c != EOF ) {
  putchar( tolower(c) );
 c = getchar();
 }
 return 0;
}
```

13

ctype.h

- Defines the functions that operate on single-byte characters
  - `tolower( )`
  - `toupper( )`
  - `islower( )`
  - `isupper( )`
  - `isalpha( )`
  - `isalnum( )`
  - `etc.`

14

Example: more compact code

```
#include <stdio.h>
#include <ctype.h>

main() /* convert input to lower case*/
{
 int c;
 while (( c = getchar() ) != EOF )
  putchar( tolower(c) );
 return 0;
}
```

15

getchar() Behaviours

- getchar() buffers input characters until a new line or EOF is entered, at which point if there is an output function executed, the buffer is written out and then emptied.
- If the buffer is currently empty and EOF is entered, the program terminates.

Data Types and Sizes in C

4 basic types in C:

- **char** – characters (8 bits)
- **int** — integers (either 16 or 32 bits)
- **float** – single precision floating point numbers (4 bytes)
- **double** – double precision floating point numbers (8 bytes)

17

Strings

A string = a one-dimensional array of characters ended by a null character '\0'

```
char greeting[6] = {'H','e','l','l','o','\0'};
char greeting[ ] = {'H','e','l','l','o','\0'};
char greeting[6] = "Hello";
char greeting[ ] = "Hello";
```

Using pointers (later):

```
char *greeting;
```

Note: must allocate space before initializing the string with "Hello"

18

printf( ) and scanf( )

```
Include file stdio.h
printf("This is a test %d \n", x);
scanf("%x %d", &x, &y);
```

%c	char
%d	int (decimal)
%f	float
%lf	double
%x	hexadecimal integer
%s	string

19

printf( ) (7.2)

```
int printf(char *format, arg1, arg2, ...);
```

- converts, formats, and prints its arguments on the standard output under control of the **format**.
- returns the number of characters printed (usually we are not interested in the returned value).

20

Examples of printf( )

```c
#include<stdio.h>

main()
{
    int a = 15, b = a / 2;
    float c = 15.3, d = c / 3;

    printf("%d\n",b);
    printf("%3d\n",b);
    printf("%03d\n",b);

    printf("%f\n",d);
    printf("%3.2f\n",d);
    printf("%8.2f\n",d);
    printf("%08.2f\n",d);
}
```

```
indigo 118 % a.out
7
  7
007
5.100000
5.10
    5.10
00005.10
```

21

Output Formatting with printf( )

- A **minus sign**, which specifies **left adjustment** of the converted argument.
- A number that specifies the **minimum field width**. The converted argument will be printed in **a field at least this wide**.
  - If necessary it will be padded on the left (or right, if left adjustment is called for) to make up the field width.
- A **period**, which **separates the field width from the precision**.
- A number, the **precision**, that specifies
  - the **maximum number of characters** to be printed from **a string**,
  - the **number of digits after the decimal point** of a **floating**-point value,
  - the **minimum number of digits** for an **integer**.

22

printf( ) with Floating-point Values

```
#include<stdio.h>

main()
{
        double d = 890.1234567;

        printf( ":%lf:\n", d );          :890.123457:
        printf( ":%10lf:\n", d );        :890.123457:
        printf( ":%.7lf:\n", d );        :890.1234567:
        printf( ":%15lf:\n", d );        :     890.123457:
        printf( ":%-15lf:\n", d );       :890.123457     :
        printf( ":%.15lf:\n", d );       :890.123456700000020:
        printf( ":%15.4lf:\n", d );      :       890.1235:
        printf( ":%-15.4lf:\n", d );     :890.1235       :
}
```

23

printf( ) Examples with Strings

```
printf(":%s:",          "hello, world");
printf(":%10s:",        "hello, world");
printf(":%.10s:",       "hello, world");
printf(":%-10s:",       "hello, world");
printf(":%.15s:",       "hello, world");
printf(":%-15s:",       "hello, world");
printf(":%15.10s:",     "hello, world");
printf(":%-15.10s:",    "hello, world");
```

```
 :%s:           :hello, world:
 :%10s:         :hello, world:
 :%.10s:        :hello, wor:
 :%-10s:        :hello, world:
 :%.15s:        :hello, world:
 :%-15s:        :hello, world   :
 :%15.10s:      :     hello, wor:
 :%-15.10s:     :hello, wor     :
```

24

printf Conversions

Character	Argument type; Printed As
d, i	int; decimal number
o	int; unsigned octal number (without a leading zero)
x, X	int; unsigned hexadecimal number (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ...,15.
u	int; unsigned decimal number
c	int; single character
s	char *; print characters from the string until a '\0' or the number of characters given by the precision.
f	double; [-]*m.dddddd*, where the number of *d*'s is given by the precision (default 6).
e, E	double; [-]*m.ddddde+/-xx* or [-]*m.ddddd*E+/-*xx*, where the number of *d*'s is given by the precision (default 6).
g, G	double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.
p	void *; pointer (implementation-dependent representation).
%	no argument is converted; print a %

25

scanf( ) (7.4)

- scanf( ) is the input analog of printf( ).
- To read an integer:

```
int num;
scanf("%d", &num);
```

- **&num** is a pointer to **num**.
- To read a char and a float:

```
char ch; float fnum;
scanf("%c %f", &ch, &fnum);
```

26

scanf( )

```
int scanf(char *format, arg1, arg2, ...);
```

- reads characters from the standard input, interprets them according to the specification in `format`, and stores the results through the remaining arguments.
- stops when it exhausts its format string, or when some input fails to match the control specification.
- returns the number of successfully matched and assigned input items (e.g., to decide how many items were found).
- returns 0 if the next input character does not match the first specification in the format string (i.e., an error).
- On the end of file, EOF is returned.
- **Note: `arg1, arg2, ...` must be pointers!**

27

scanf Conversions

Character	Input Data; Argument type
d	decimal integer; `int *`
i	integer; `int *`. The integer may be in octal (leading `0`) or hexadecimal (leading `0x` or `0X`).
o	octal integer (with or without leading zero); `int *`
u	unsigned decimal integer; `unsigned int *`
x	hexadecimal integer (with or without leading `0x` or `0X`); `int *`
c	characters; `char *`. The next input characters (default 1) are placed at the indicated spot. The normal skip-over white space is suppressed; to read the next non-white space character, use `%1s`
s	character string (not quoted); `char *`, pointing to an array of characters long enough for the string and a terminating `'\0'` that will be added.
e,f,g	floating-point number with optional sign, optional decimal point and optional exponent; `float *`
%	literal %; no assignment is made.

28

I/O Redirection

`prog < infile`

- prog reads characters from infile instead of the standard input.

`prog > outfile`

- prog writes to outfile instead of the standard output.

`otherprog | prog`

- Output from otherprog is the input to prog.

`prog | anotherprog`

- puts the standard output of prog into the standard input of anotherprog.

29

Next Lecture

- Types, Operators and Expressions (Chapter 2)

30