Week 3 Lecture 2

Types Constants and Variables

Types

- Computers store bits: strings of 0s and 1s
- Types define how bits are interpreted
  - They can be integers (whole numbers): 1, 2, 3
  - They can be characters 'a', '?', '_'
  - They can be real numbers: 1.5, 3.2453
  - They can be strings "Hello World"
- In every case they are strings of bits.
  - Scratch figures out what the bits mean by what you do with them

Types in C

- In C you must tell the compiler how to interpret the bits
- There are two kinds of types:
  - Simple types, which are defined by the hardware
  - Complex types, which are define by the language
    - Complex types are collections of simple types

Simple Types

- There are three basic simple types: int, char, float
  - Int: an integer
    - E.g., 1, 2, 3
  - Char: a character
    - E.g., 'a', '/', '_'
  - Float: a real number
    - E.g., 1.5, 2.354

Types: Size and Range

Name	**Description**	**Size***	**Range***
char	Character or small integer	1 byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short integer	2 bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
float	Floating point number	4 bytes	3.4e +/- 38 (7 digits)
double	Double precision floating point number	8 bytes	1.7e +/- 308 (15 digits)
long double	Long double precision floating point number	8 bytes	1.7e +/- 308 (15 digits)

Fixed point number

- Binary number
- Sign bit
  - Two's compliment
- Limited numbers
  - [-214748646, 214748647]

Floating Point Numbers

$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10}_{\text{base}}{}^{\overbrace{-4}^{\text{exponent}}}$$

- Represents fractions
- Represents large and small numbers
- Have limited precision
  - 23 bits of significand
    - up to about 8 decimal digits
  - 8 bits of exponent
    - About plus or minus: 2^{256}
      - 115792089237316195423570985008687907853269984665640564039457584007913129639936
  - 1 sign bit

Characters and Strings

- Characters are 8 bit values interpreted as ASCII.
- Strings a sequences and char's terminated by the null character '/0'.
  - Strings are complex data types. Their length cannot be specified in the standard.

Constants

- Constants are representations of value of certain types.
  - Int constant: 123
    - Comprised entirely of digits
  - Float constant: 1.234
    - Includes a period, but otherwise only digits
  - Char constant: 'a'
    - A single character, surrounded by single quotes
  - String constant: “Hello world”
    - Zero or more characters surrounded by double quotes

Variables

- Variables are named memory locations of a size that can hold values of a certain type.
  - Int variable: int counter;
  - Float variable: float average;
  - Char variable: char operator;
  - String variable: char message[80];
    - The [80] indicates that this string is 79 characters or less.

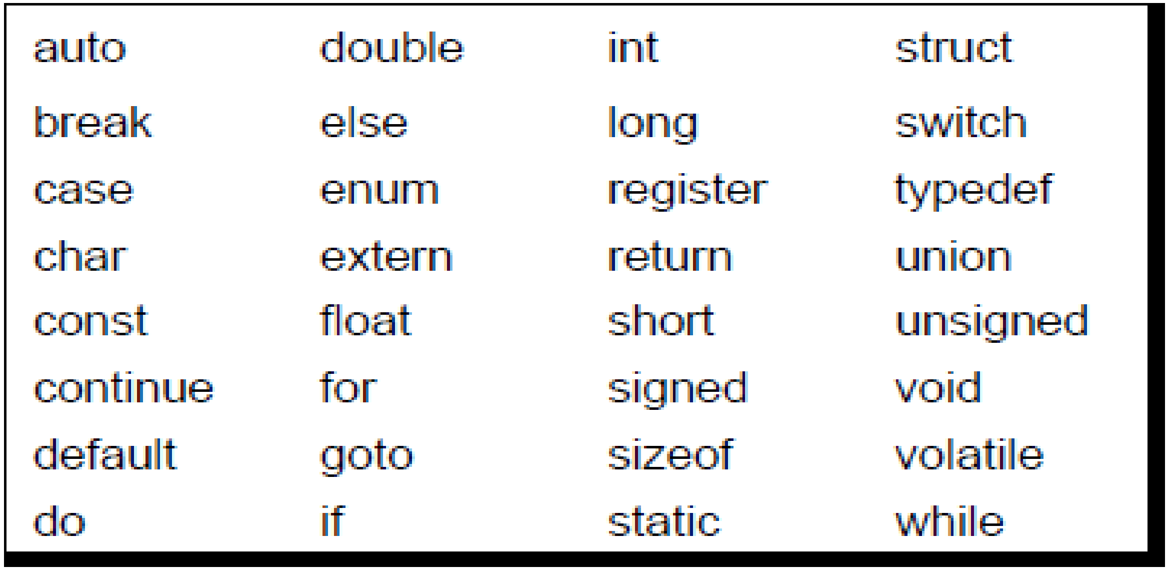
Rules for Constructing Variables Names

- A variable name is any combination of letters, digits and underscores
- The first character must be an letter or an underscore (system variable).
- No commas and blanks are allowed within a variable name.
- Case matters!
- C keywords cannot be be used as variable names.
- Examples:

```
present, hello, y2x3, r2d3, ...        /* OK */
_1993_tar_return                       /* system var */
Hello#there                            /* illegal: # */
double                                 /* keyword */
2fartogo                               /* illegal: 2 */
```

Keywords

- Keywords, or reserved words, are defined by C
  - They cannot be used as variable names
- There are 32:



auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Initialization

- Always initialize variables with constants.
  - Int variable: int counter = -1;
  - Float variable: float average = -1.0;
  - Char variable: char operator = 'x';
  - String variable:
    - char message[80] = “Initialization”;

Initialization Tips

- If possible, initialize to a value that doesn't make sense so you know when you are seeing the initial value.
  - When initializing a string “initialization value” is a good choice.
    - On the other hand, sometimes you want to initialize to a default value
  - If a numeric results is supposed to be positive -1 or -1.1 is a good choice.
  - A character you do not expect is a good choice.

Compound types

- Arrays: all elements are the same size.
  - e.g. int array[5]
    - An array containing 5 integers.
  - To use: array[0] = 5;
- Structs: elements may have different sizes
  - e.g. struct circle {int x, y; float radius}
    - A circle has two ints called x and y and a single float called radius.
  - To use: circle.x = 5;

Strings are arrays.

- E.g. char *Name = “Nat”;
  - Name[0] == 'N'
  - Name[1] = 'a'
  - Name[2] = 't'
  - Name[3] = '\0'

Arrays are pointers

- e.g. char* Name = “Nat”
  - Name == &Name[0]
  - *Name == Name[0] == 'N'
  - &Name[1] == Name + sizeof(char)
  - &Name[2] == Name + (2 * sizeof(char))

Array initialization

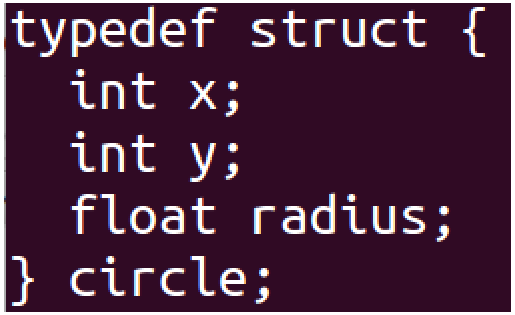
- Arrays can be initialized.
  - e.g. int data[5] = {2, 5, 3, 7, 1}

Structs are pointers

- e.g. struct circle {int x, y; float radius}
  - &circle.x == circle
  - circle.x == *circle
  - &circle.y == circle + sizeof(int)
  - &circle.radius == circle + (2 *sizeof(int))

Define new types with typedef

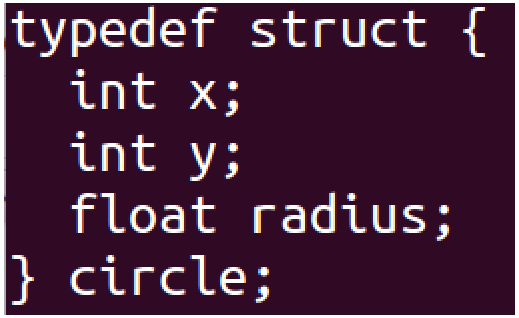
- Type definition common with struct
- Define a circle type
  - Ints x and y
  - Float radius
- Allocates memory for each of the elements.



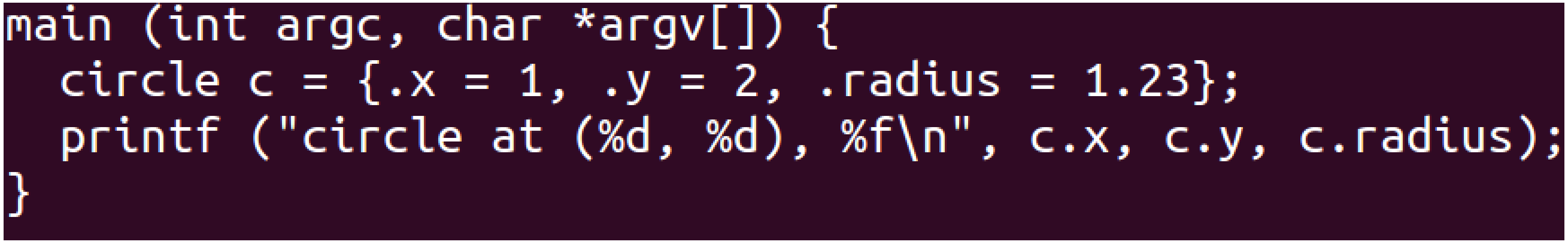
```
typedef struct {
  int x;
  int y;
  float radius;
} circle;
```

Struct initialization

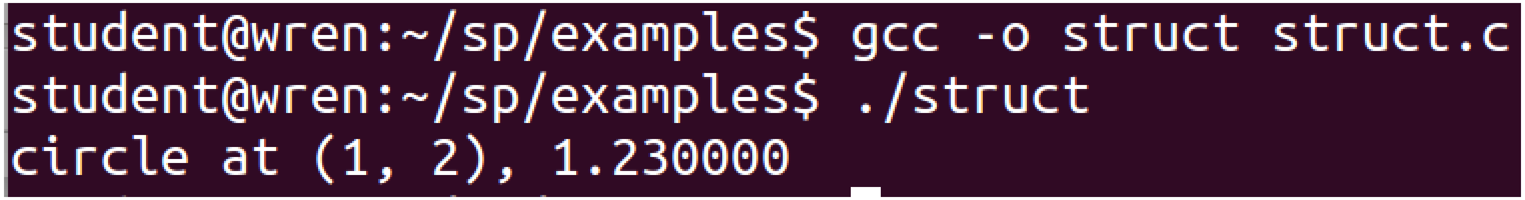
- Structs can be initialized.
- Easiest with defined type



```
typedef struct {
  int x;
  int y;
  float radius;
} circle;
```



```
main (int argc, char *argv[]) {
  circle c = {.x = 1, .y = 2, .radius = 1.23};
  printf ("circle at (%d, %d), %f\n", c.x, c.y, c.radius);
}
```



```
student@wren:~/sp/examples$ gcc -o struct struct.c
student@wren:~/sp/examples$ ./struct
circle at (1, 2), 1.230000
```

Kinds of variables

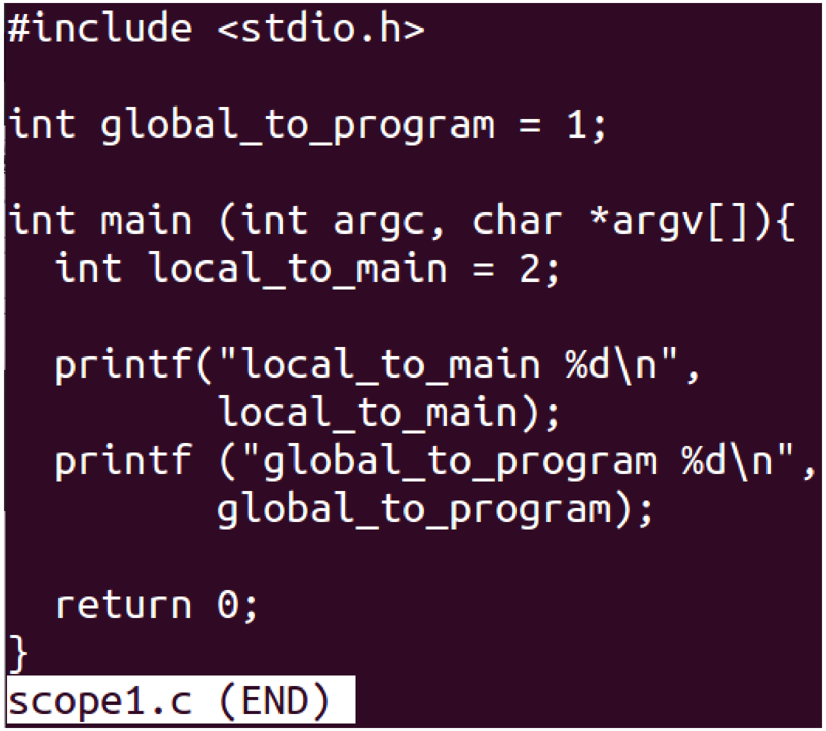
- **Local variable**: defined in the current block of code.
- **Global variable**: defined through the program.
- **Static variable**: allocated memory survives though program life.
- **External variable**: variable is defined in another file.

Scope and Lifetime

- Scope defines where a variable can be used
  - Local variable: used only in block.
  - Global variable: used anywhere in program.
- Lifetime defines how long a variable exists.
  - Local variables: usually deleted when not accessible
  - Static variables: exists until program ends.

Scope example

- Global variable
  - Can be used anywhere in program.
  - Avoid
    - Difficult to find where it changes
- Local variable
  - Can only be used in current block.



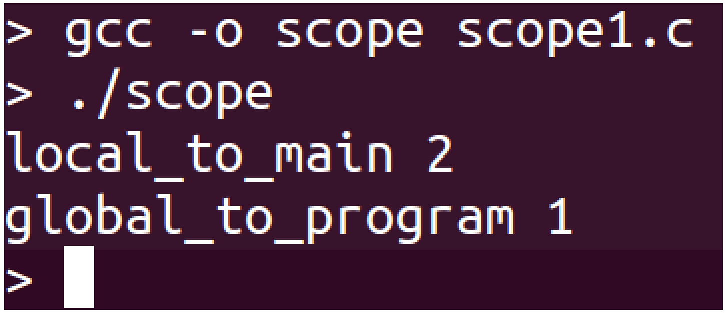
```
#include <stdio.h>

int global_to_program = 1;

int main (int argc, char *argv[]){
  int local_to_main = 2;

  printf("local_to_main %d\n",
         local_to_main);
  printf ("global_to_program %d\n",
         global_to_program);

  return 0;
}
scope1.c (END)
```



```
> gcc -o scope scope1.c
> ./scope
local_to_main 2
global_to_program 1
> 
```