Week 1 Lecture 2

NWEN 241
Systems Programming

Alvin C. Valera

alvin.valera@ecs.vuw.ac.nz

Content

C Fundamentals

- Identifiers and reserved keywords
- Data types, constants and literals
- Operators and expressions
- Flow control
- Functions

Identifiers

- Identifier is used to name **macros**, variables, **functions**, **structs**, **unions**, and other entities in a computer program

- Java and C have similar rules for identifiers, except:
  - In C, $ is not allowed in identifiers (though some compilers allow $)

Rules on Identifiers

- An identifier is a sequence of letters and digits
  - The first character must be a letter
    - The underscore character _ counts as a letter
    - Upper and lower case letters are different
- Identifiers may have any length
  - Usually, only the first 31 characters are significant
  - For macro names, only the first 63 characters are significant
- Reserved keywords cannot be used as identifiers!

Examples

`counter`	Valid: consists of letters
`_Temp_variable_2`	Valid: consists of letters and digits
`1myVariable`	Invalid: first character is not a letter
`$steps`	Invalid: $ is not allowed in C
`continue`	Invalid: reserved word

Reserved Keywords

```
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
```

Data Types

- Recall: Java has 8 basic data types which have fixed sizes

Data Type	Size (bytes)
`boolean`	1
`byte`	1
`char`	2
`short`	2
`int`	4
`long`	8
`float`	4
`double`	8

Data Types

- C data types:

Data Type	Size (bytes)
~~boolean~~	~~1~~
~~byte~~	~~1~~
char	~~2~~ 1
short (short int)	~~2~~ Machine-dependent
int	~~4~~ Machine-dependent
long (long int)	~~8~~ Machine-dependent
long long (long long int)	Machine-dependent
float	~~4~~ Machine-dependent
double	~~8~~ Machine-dependent
long double	10

Integral types

Float types

Data Type Size

- Sizes of different types
  - Use `sizeof()` to find out
  - Some of the types size may vary from machine to machine

- The following rules are always guaranteed:
  - `sizeof(char) == 1`
  - `sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)`
  - `sizeof(float) <= sizeof(double) <= sizeof(long double)`

Data Types

- Integral types can either be `signed` or `unsigned`

```
signed int var1;    // Signed integer

unsigned int var2;  // Unsigned integer

int var1;  // If signed or unsigned is not present, default is signed
```

char Data Type

- `unsigned char`: 0 to 255; `signed char`: -128 to 127
- `char` is meant to hold **1 ASCII character**

0	NUL	1	SOH	2	STX	3	ETX	4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	NL	11	VT	12	NP	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3	20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC	28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	DEL

Variable Declaration

- Similar syntax as Java
- A variable must be declared before it can be used
- A variable may be initialized in its declaration
  - If variable name is followed by an equals sign and an expression, the latter serves as an *initializer*

```
int i = 0, j = 1, k = 2;
char c = 'A';
float f = 1.25;
```

- Possible initializers
  - Constant
  - Expression

Constants and Literals

- Constants are **fixed values** that cannot be changed during a program's execution
- The fixed values are called **literals**
- Literals
  - Integer
  - Floating Point
  - Character
  - *String*
  - *Enumeration*

Integer Literals

- Used for representing integer-valued constants
  - Can be written in decimal (no prefix), octal (prefix `0`), or hexadecimal (prefix `0x`)
  - Can have suffix that is a combination of `U` (`unsigned`) and `L` (`long`) in any order
    - No suffix means the literal is of type `int`

`12345`	Valid
`12345u`	Valid: unsigned
`0xbeef`	Valid: hexadecimal
`081`	Invalid: 8 is not a valid octal digit
`0x123uu`	Invalid: same suffix is repeated

Floating Point Literals

- Used for representing real-valued constants
  - Can be written in decimal form or exponential form
  - Can have suffix `f` (`float`) or `L` (`long double`)
    - No suffix means the literal is of type **`double`**

`3.1415`	Valid (decimal form)
`31415e-4`	Valid (exponential form)
`31415e-4L`	Valid: long double
`6.22e`	Invalid: incomplete exponent
`.e23`	Invalid: missing decimal/fraction part

Character Literals

- Used for representing character constants
  - Enclosed in single quotes (')
  - Can be plain (single character) or escape (single character preceded by \)

`'A'`	Valid (plain character)
`'\t'`	Valid (escape character): tab
`'Aa'`	Invalid: multiple characters in single quotes
`'\z'`	Invalid: not a valid escape character

Declaring Constants

- Constants can be declared using `const` qualifier or `#define` pre-processor
- Such named constants are also called **symbolic constants**

```
const float PI = 3.14;
const int MAX = 12345;
```

```
#define PI 3.14
#define MAX 12345
```

Type Casting

- Type casting is a way to convert a variable from one data type to another data type
- C performs automatic type casting (implicit type conversion)

```
int i = 2;
double d = 2.5;
i = (int)d;       // explicit type casting

i = d;            // d is converted to an int
                  // and then assigned to i
```

Operators

- Java and C share many of the built-in operators
  - Arithmetic
  - Assignment
  - Increment/decrement
  - Relational
  - Equality and logical
  - Bitwise
- C specific operators
  - Pointers and reference related operators (*, &, ->)
  - Others (sizeof, scope, casting)

Operator Precedence

- Operator *precedence* determines the sequence in which operators in an expression are evaluated

- *Associativity* determines execution for operators of equal precedence

- Precedence can be overridden by explicit grouping using parenthesis: (and)

Operator Precedence Table (not complete)

Unary operators

Arithmetic operators

Ternary operator

Assignment operators

Operators	Associativity
`() [] -> .`	left to right
`! ~ ++ -- + - * (type) sizeof`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >>`	left to right
`< <= > >=`	left to right
`== !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= |= <<= >>=`	right to left
`,`	left to right

Important Things to Remember

- `/` denotes integer division if both operands are of integral types
  - `5/2` evaluates to 2 (integer part is used, decimal part is truncated)

- `%` denotes modulo operation
  - `5%2` evaluates to 1 (the remainder after dividing 5 with 2)

- Increment/decrement operators can only be applied to variables of basic types

```
k++;
counter--;
```

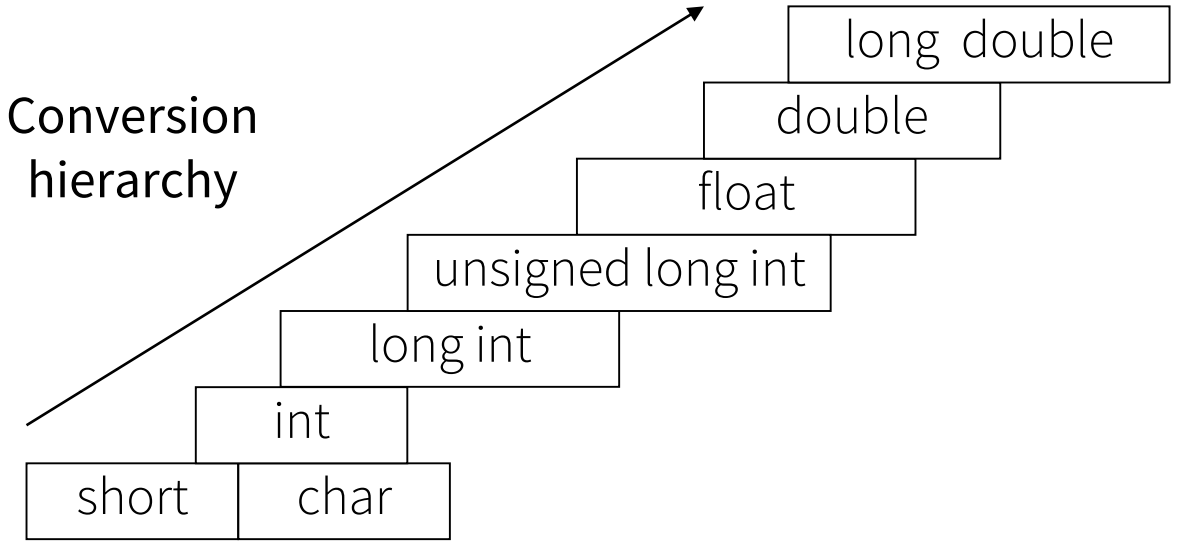
Valid if `k` and `counter` are variables of basic types

```
777++;
(a + b*c)--;
```

Invalid

“Conversion hierarchy”

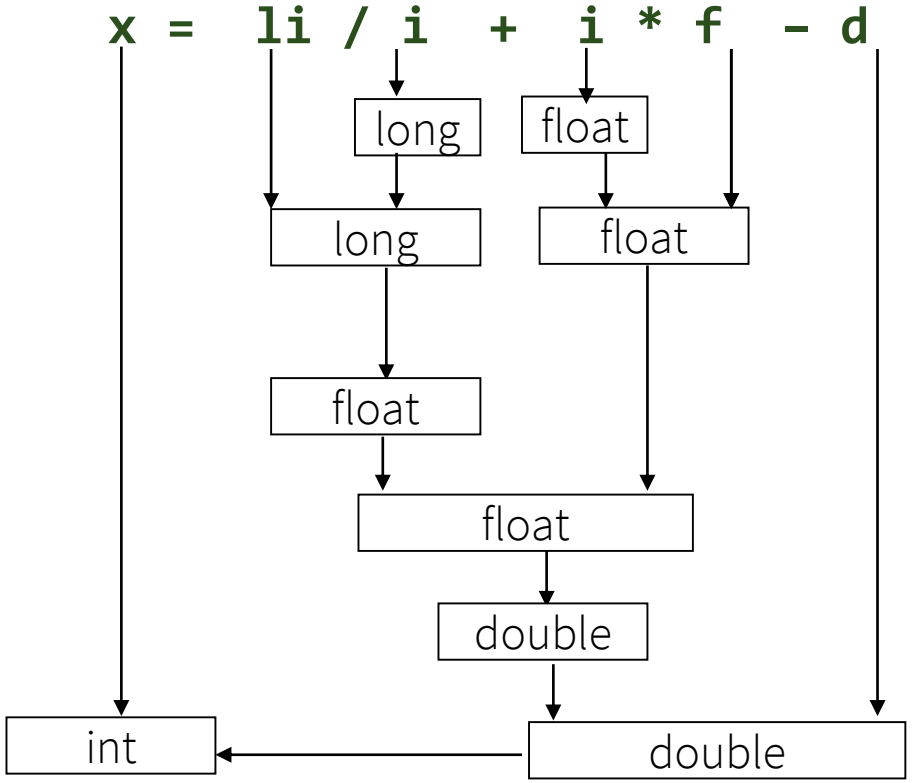
- What happens when operands have different types in an arithmetic expression?
  - **Implicit type conversion is performed:** compiler automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any significance



Implicit Type Conversion Example

Suppose:

```
int i, x;
float f;
double d;
long int li;
```



The final result of the right hand side expression is converted to the type of the variable on the left of the assignment

Control Constructs

- Control flow
  - If-else
  - Else-if
  - Switch
- Loop
  - While-loop
  - For-loop
  - Do-while-loop
- Same syntax as Java

Differences

Condition in if-else, else-if, while-loop, for-loop and do-while-loop

- In Java, the condition must be an expression that evaluates to boolean
- In C, the condition is an expression that evaluates to any type
  - Considered true if expression evaluates to non-zero value, otherwise false

Break and continue

- In Java, break and continue statements can be labelled or unlabelled
- In C, break and continue statements do not support labels

Example

```
int i = 100;

while (i--) {
   // do stuff
}
```

- Valid in C
- Will generate syntax error in Java
  - Condition inside while-loop should be changed to an expression that will evaluate to boolean type, e.g. `i-- > 0`

Functions

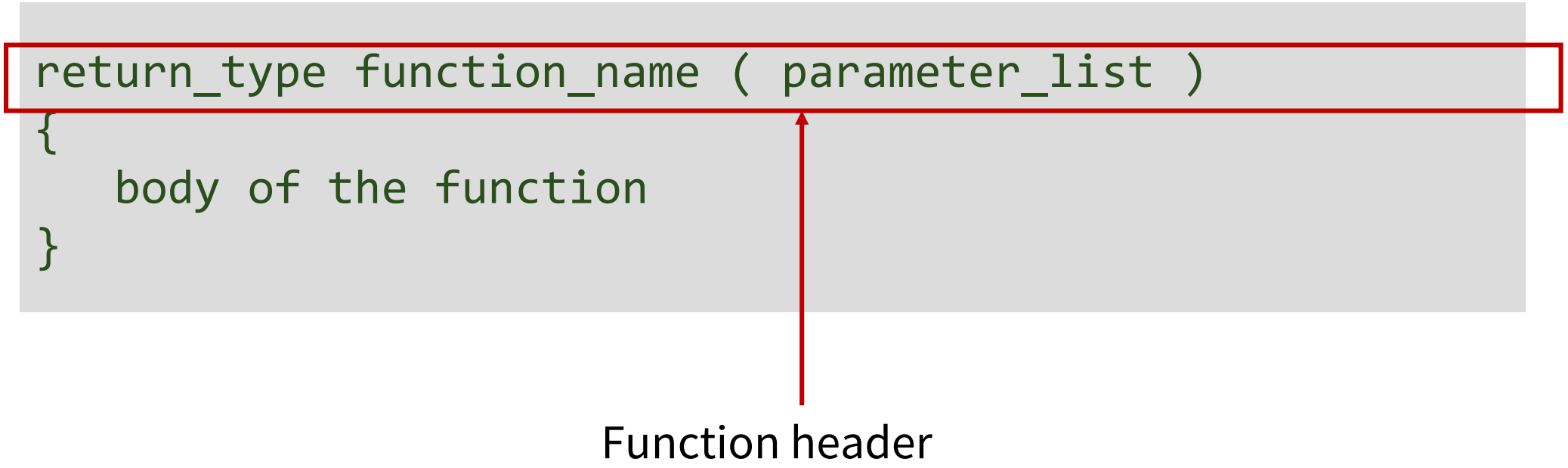
- Unlike Java, C allows functions to exist on their own, i.e., outside any class
  - In C, functions are first-class entities: a C program consists of one or more functions

- A C program must have exactly one `main` function
- Execution begins with the `main` function

Functions

- General form of a C **function definition:**

```
return_type function_name ( parameter_list )
{
   body of the function
}
```

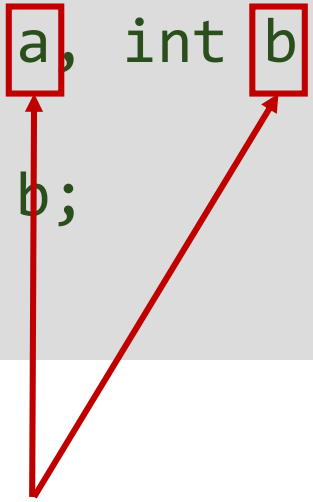


Functions

- Examples

```
void say_hello ( void )
{
   printf("Hello");
}
```

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



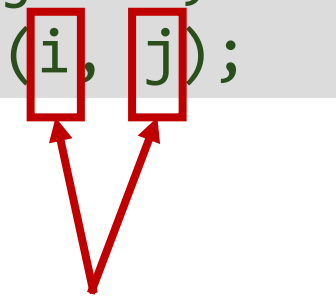
Formal parameters

Invoking Functions

- Example function invocations:

```
say_hello();
```

```
int i = 1, j = 2;
int k = add(i, j);
```



Actual parameters

- Before a function can be invoked, either the **function definition** or **function prototype** should have been declared prior to the invocation

Function Prototype

- A declaration specifying the return type, function name, and list of parameter types

```
return_type function_name ( parameter_types_list );
```

Function Prototype

- Examples

```
void say_hello ( void );
```

```
int add ( int a, int b );
```

- No need to provide identifiers to input parameters, the types of the input parameters are sufficient

```
int add ( int, int );
```

Next Lecture

- Function-Like Macros
- Arrays