Princeton University
COS 217: Introduction to Programming Systems
C Primitive Data Types

Type: **int** (or **signed int**)

Description: A (positive or negative) integer.

Size: System dependent. On armlab with gcc217: 4 bytes.

Example Variable Declarations:

```
int iFirst;
signed int iSecond;
```

Example Literals (assuming size is 4 bytes):

C Literal	Binary Representation	Note
123	00000000 00000000 00000000 01111011	decimal form
-123	11111111 11111111 11111111 10000101	negative form
0173	00000000 00000000 00000000 01111011	octal form
0x7B	00000000 00000000 00000000 01111011	hexadecimal form
2147483647	01111111 11111111 11111111 11111111	largest
-2147483648	10000000 00000000 00000000 00000000	smallest

Type: **unsigned int**

Description: A non-negative integer.

Size: System dependent. sizeof(unsigned int) == sizeof(int). On armlab with gcc217: 4 bytes.

Example Variable Declaration:

```
unsigned int uiFirst;
unsigned uiSecond;
```

Example Literals (assuming size is 4 bytes):

C Literal	Binary Representation	Note
123U	00000000 00000000 00000000 01111011	decimal form
0173U	00000000 00000000 00000000 01111011	octal form
0x7BU	00000000 00000000 00000000 01111011	hexadecimal form
4294967295U	11111111 11111111 11111111 11111111	largest
0U	00000000 00000000 00000000 00000000	smallest

Type: **long** (or **long int** or **signed long** or **signed long int**)

Description: A (positive or negative) integer.

Size: System dependent. sizeof(long) >= sizeof(int). On armlab with gcc217: 8 bytes.

Example Variable Declarations:

```
long lFirst;
long int iSecond;
signed long lThird;
signed long int lFourth;
```

Example Literals (assuming size is 8 bytes):

C Literal	Binary Representation/Note
123L	00000000 00000000 00000000 00000000 00000000 00000000 00000000 01111011 decimal form
-123L	11111111 11111111 11111111 11111111 11111111 11111111 11111111 10000101 negative form
0173L	00000000 00000000 00000000 00000000 00000000 00000000 00000000 01111011 octal form
0x7BL	00000000 00000000 00000000 00000000 00000000 00000000 00000000 01111011 hexadecimal form
9223372036854775807L	01111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 largest
-9223372036854775808L	10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 smallest

---

Type: unsigned long (or **unsigned long int**)

Description: A non-negative integer.

Size: System dependent. sizeof(unsigned long) == sizeof(long). On armlab with gcc217: 8 bytes.

Example Variable Declarations:

```
unsigned long ulFirst;
unsigned long int ulSecond;
```

Example Literals (assuming size is 8 bytes):

C Literal	Binary Representation/Note
123UL	00000000 00000000 00000000 00000000 00000000 00000000 00000000 01111011 decimal form
0173UL	00000000 00000000 00000000 00000000 00000000 00000000 00000000 01111011 octal form
0x7BUL	00000000 00000000 00000000 00000000 00000000 00000000 00000000 01111011 hexadecimal form
18446744073709551615UL	11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 largest
0UL	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 smallest

---

Type: signed char

Description: A (positive or negative) integer. Usually represents a character according to a character code (e.g., ASCII).

Size: 1 byte.

Example Variable Declarations:

```
signed char cSecond;
```

Example Literals (assuming the ASCII code is used):

C Literal	Binary Representation	Note
(signed char)'a'	01100001	character form
(signed char)97	01100001	decimal form
(signed char)0141	01100001	octal form
(signed char)0x61	01100001	hexadecimal form
(signed char)'\o141'	01100001	octal character form
(signed char)'\x61'	01100001	hexadecimal character form
(signed char)123	01111011	decimal form
(signed char)-123	10000101	negative form

(signed char)127	01111111	largest
(signed char)-128	10000000	smallest
(signed char)'\0'	00000000	the null character
(signed char)'\a'	00000111	bell
(signed char)'\b'	00001000	backspace
(signed char)'\f'	00001100	formfeed
(signed char)'\n'	00001010	newline
(signed char)'\r'	00001101	carriage return
(signed char)'\t'	00001001	horizontal tab
(signed char)'\v'	00001011	vertical tab
(signed char)'\\'	01011100	backslash
(signed char)'\''	00100111	single quote

Type: unsigned char

Description: A non-negative integer. Usually represents a character according to a character code (e.g., ASCII).

Size: 1 byte.

Example Variable Declaration:

```
unsigned char ucFirst;
```

Example Literals (assuming the ASCII code is used):

C Literal	Binary Representation	Note
(unsigned char)'a'	01100001	character form
(unsigned char)97	01100001	decimal form
(unsigned char)0141	01100001	octal form
(unsigned char)0x61	01100001	hexadecimal form
(unsigned char)'\o141'	01100001	octal character form
(unsigned char)'\x61'	01100001	hexadecimal character form
(unsigned char)123	01111011	decimal form
(unsigned char)255	11111111	largest
(unsigned char)0	00000000	smallest
(unsigned char)'\0'	00000000	the null character
(unsigned char)'\a'	00000111	bell
(unsigned char)'\b'	00001000	backspace
(unsigned char)'\f'	00001100	formfeed
(unsigned char)'\n'	00001010	newline
(unsigned char)'\r'	00001101	carriage return
(unsigned char)'\t'	00001001	horizontal tab
(unsigned char)'\v'	00001011	vertical tab
(unsigned char)'\\'	01011100	backslash
(unsigned char)'\''	00100111	single quote

Type: char

Description:

On some systems "char" is the same as "signed char".
On some systems "char" is the same as "unsigned char".
On armlab with gcc217 "char" is the same as "unsigned char".

Type: short (or **short int**, or **signed short**, or **signed short int**)

Description: A (positive or negative) integer.

Size: System dependent. sizeof(short) <= sizeof(int). On armlab with gcc217: 2 bytes.

Example Variable Declarations:

```
short sFirst;
short int sSecond;
```

```
signed short sThird;
signed short int sFourth;
```

Example Literals (assuming size is 2 bytes):

C Literal	Binary Representation	Note
(short)123	00000000 01111011	decimal form
(short)-123	11111111 10000101	negative form
(short)32767	01111111 11111111	largest
(short)-32768	10000000 00000000	smallest
(short)0173	00000000 01111011	octal form
(short)0x7B	00000000 01111011	hexadecimal form

Type: unsigned short (or **unsigned short int**)

Description: A non-negative integer.

Size: System dependent. sizeof(unsigned short) == sizeof(short). On armlab with gcc217: 2 bytes.

Example Variable Declarations:

```
unsigned short usFirst;
unsigned short int usSecond;
```

Example Literals (assuming size is 2 bytes):

C Literal	Binary Representation	Note
(unsigned short)123	00000000 01111011	decimal form
(unsigned short)0173	00000000 01111011	octal form
(unsigned short)0x7B	00000000 01111011	hexadecimal form
(unsigned short)65535	11111111 11111111	largest
(unsigned short)0	00000000 00000000	smallest

Type: double

Description: A (positive or negative) double-precision floating point number.

Size: System dependent. On armlab with gcc217: 8 bytes.

Example Variable Declaration:

```
double dFirst;
```

Example Literals (assuming size is 8 bytes):

C Literal	Note
123.456	fixed-point notation
1.23456E2	scientific notation
.0123456	fixed-point notation
1.234546E-2	scientific notation with negative exponent
-123.456	fixed-point notation
-1.23456E2	scientific notation with negative mantissa
-.0123456	fixed-point notation
-1.23456E-2	scientific notation with negative mantissa and negative exponent
1.797693E308	largest (approximate)
-1.797693E308	smallest (approximate)
2.225074E-308	closest to 0 (approximate)

Type: float

Description: A (positive or negative) single-precision floating point number.

Size: System dependent. sizeof(float) <= sizeof(double). On armlab with gcc217: 4 bytes.

Example Variable Declaration:

```
float fFirst;
```

Example Literals (assuming size is 4 bytes):

C Literal	Note
123.456F	fixed-point notation
1.23456E2F	scientific notation
.0123456F	fixed-point notation
1.234546E-2F	scientific notation with negative exponent
-123.456F	fixed-point notation
-1.23456E2F	scientific notation with negative mantissa
-.0123456F	fixed-point notation
-1.23456E-2F	scientific notation with negative mantissa and negative exponent
3.402823E38F	largest (approximate)
-3.402823E38F	smallest (approximate)
1.175494E-38F	closest to 0 (approximate)

Type: long double

Description: A (positive or negative) extended-precision floating point number.

Size: System dependent. sizeof(long double) >= sizeof(double). On armlab with gcc217: 16 bytes.

Example Variable Declaration:

```
long double ldFirst;
```

Example Literals (assuming size is 16 bytes):

C Literal	Note
123.456L	fixed-point notation
1.23456E2L	scientific notation
.0123456L	fixed-point notation
1.234546E-2L	scientific notation with negative exponent
-123.456L	fixed-point notation
-1.23456E2L	scientific notation with negative mantissa
-.0123456L	fixed-point notation
-1.23456E-2L	scientific notation with negative mantissa and negative exponent
1.18973E4932L	largest (approximate)
-1.189731E4932L	smallest (approximate)
3.3621E-4932L	closest to 0 (approximate)

Differences between C and Java:

Java only:
boolean, byte

C only:
unsigned char, unsigned short, unsigned int, unsigned long
long double

Java: Sizes of all types are **specified**
C: Sizes of all types except char are **system dependent**

Java: char comprises **2** bytes
C: char comprises **1** byte

sizes.c (Page 1 of 1)

```
 1: /*-------------------------------------------------------------------------*/
 2: /* sizes.c                                                                 */
 3: /* Author: Bob Dondero                                                     */
 4: /*-------------------------------------------------------------------------*/
 5:
 6: #include <stdio.h>
 7:
 8: /*  Write the size, in bytes, of each fundamental data type
 9:     to stdout. Return 0. */
10:
11: int main(void)
12: {
13:     printf("Bytes per char:            %d\n",
14:        (int)sizeof(char));
15:     printf("Bytes per unsigned char:   %d\n",
16:        (int)sizeof(unsigned char));
17:     printf("Bytes per short:           %d\n",
18:        (int)sizeof(short));
19:     printf("Bytes per unsigned short:  %d\n",
20:        (int)sizeof(unsigned short));
21:     printf("Bytes per int:             %d\n",
22:        (int)sizeof(int));
23:     printf("Bytes per unsigned int:    %d\n",
24:        (int)sizeof(unsigned int));
25:     printf("Bytes per long:            %d\n",
26:        (int)sizeof(long));
27:     printf("Bytes per unsigned long:   %d\n",
28:        (int)sizeof(unsigned long));
29:     printf("Bytes per size_t:          %d\n",
30:        (int)sizeof(size_t));
31:     printf("Bytes per float:           %d\n",
32:        (int)sizeof(float));
33:     printf("Bytes per double:          %d\n",
34:        (int)sizeof(double));
35:     printf("Bytes per long double:     %d\n",
36:        (int)sizeof(long double));
37:     printf("Bytes per pointer:         %d\n",
38:        (int)sizeof(void*));
39:
40:     return 0;
41: }
42:
43: /* Example execution:
44:
45: $ gcc217 sizes.c -o sizes
46:
47: $ ./sizes
48: Bytes per char:            1
49: Bytes per unsigned char:   1
50: Bytes per short:           2
51: Bytes per unsigned short:  2
52: Bytes per int:             4
53: Bytes per unsigned int:    4
54: Bytes per long:            8
55: Bytes per unsigned long:   8
56: Bytes per size_t:          8
57: Bytes per float:           4
58: Bytes per double:          8
59: Bytes per long double:     16
60: Bytes per pointer:         8
61: */
```

Princeton University
COS 217: Introduction to Programming Systems
C Operators

Grouped by Category:

Operator	Precedence	Category	Description	Associativity
++	2	arithmetic	Increment	R to L
--	2	arithmetic	Decrement	R to L
+	2	arithmetic	Unary positive	R to L
-	2	arithmetic	Unary negative	R to L
*	3	arithmetic	Multiplication	L to R
/	3	arithmetic	Division	L to R
%	3	arithmetic	Modulus	L to R
+	4	arithmetic	Addition	L to R
-	4	arithmetic	Subtraction	L to R
=	14	assignment	Assignment	R to L
+=	14	assignment	Addition and assignment	R to L
-=	14	assignment	Subtraction and assignment	R to L
*=	14	assignment	Multiplication and assignment	R to L
/=	14	assignment	Division and assignment	R to L
%=	14	assignment	Modulus and assignment	R to L
<	6	relational	Less than	L to R
<=	6	relational	Less than or equal to	L to R
>	6	relational	Greater than	L to R
>=	6	relational	Greater than or equal to	L to R
==	7	relational	Equality	L to R
!=	7	relational	Inequality	L to R
!	2	logical	Logical "not"	R to L
&&	11	logical	Logical "and"	L to R
||	12	logical	Logical "or"	L to R
[]	1	pointer	Array element select	L to R
*	2	pointer	Dereference	R to L
&	2	pointer	Address of	R to L
->	1	structure	Structure dereference and field select	L to R
.	1	structure	Structure field select	L to R
~	2	bitwise	Bitwise "not"	R to L
<<	5	bitwise	Bitwise shift left	L to R
>>	5	bitwise	Bitwise shift right	L to R
&	8	bitwise	Bitwise "and"	L to R
^	9	bitwise	Bitwise "exclusive or"	L to R
|	10	bitwise	Bitwise "or"	L to R
&=	14	bitwise	Bitwise "and" and assignment	R to L
^=	14	bitwise	Bitwise "exclusive or" and assignment	R to L
|=	14	bitwise	Bitwise "or" and assignment	R to L
<<=	14	bitwise	Bitwise left shift and assignment	R to L
>>=	14	bitwise	Bitwise right shift and assignment	R to L
()	1	function	Function call	L to R
(*type*)	2	cast	Cast	R to L
sizeof	2	sizeof	size of (compiletime)	R to L
?:	13	ternary	Conditional expression (ternary)	R to L
,	15	sequence	Sequence	L to R

Differences between C and Java

Java only:

`>>>`	Right shift with zero extension
`new`	Create an object
`instanceof`	Is left operand an object of class right-operand?

C only:

`->`	structure member select
`*`	dereference
`&`	address of
`,`	sequence
`sizeof`	compile-time sizeof

Related to type boolean:

Java: Relational and logical operators evaluate to type `boolean`
C: Relational and logical operators evaluate to type `int`
Java: Logical operators take operands of type `boolean`
C: Logical operators take operands of type `int`

Related to class String:

Java: Operators + and += can concatenate `String` objects
C: Operators + and += do not concatenate `String` objects – because there are no `String` objects

Java: Demotions are not automatic
C: Demotions are automatic

```
int i;
char c;
...
i = c;          /* Implicit promotion. */
                /* OK in Java and C. */

c = i;          /* Implicit demotion. */
                /* Java: Compiletime error. */
                /* C: OK.  Truncation without warning. */

c = (char)i;    /* Explicit demotion. */
                /* Java: Truncation without warning. */
                /* C: Truncation without warning. */
```

Princeton University
COS 217: Introduction to Programming Systems
C Statements

Statement Type	Statement Syntax	Examples

Expression Statement	*expression*;	i = 5;
		printf("Hello");
		5; /* valid, but nonsensical */

Declaration Statement	*modifiers datatype variable [= initialvalue][,variable [= initialvalue]]...*;	int i; int i, j; int i = 5, j = 6; const int i; static int i; extern int i;

Compound Statement (alias Block)	{ *statement statement ...* }	{
		  int i;
		  i = 5;
		  ...
		}

If Statement	if (*integerexpr*) *statement*;
	if (*pointerexpr*) *statement*;
		if (i == 5)
		{
		  statement;
		  statement;
		}

Switch Statement	switch (*integerexpr*)
	{
	  case *integerconstant*: *statements*
	  case *integerconstant*: *statements*
	  default: *statements*
	}
		switch (i)
		{
		  case 1: *statement*; break;
		  case 2: *statement*; break;
		  default: *statement*;
		}

While Statement	while (*integerexpr*) *statement*
	while (*pointerexpr*) *statement*
		while (i < 5)
		{
		  statement;
		  statement;
		}

DoWhile Statement	do *statement* while (*integerexpr*);
	do *statement* while (*pointerexpr*);
		do
		{
		  statement;
		  statement;
		} while (i < 5);

For Statement	for (*initexpr*; *integerexpr*; *increxpr*)
	  statement
	for (*initexpr*; *pointerexpr*; *increxpr*)
	  statement
		for (i = 0; i < 5; i++)
		{
		  statement;
		  statement;
		}

Return Statement	return; return *expr*;	return; return i + 5;

Break Statement	break;	while (i < 5)
		{
		  statement;
		  if (j == 6)
		  break;
		  statement;
		}

Continue Statement	continue;	while (i < 5)
		{
		  statement;
		  if (j == 6)
		  continue;
		  statement;
		}

Goto Statement	goto *label*;	mylabel:
		  ...
		  goto mylabel;
		  ...

Differences between C and Java:

Expression Statement:
- Java: Only expressions that have a side effect can be made into expression statements
- C: Any expression can be made into an expression statement
- Java: Has `final` variables
- C: Has `const` variables

Declaration Statement:
- Java: Compile-time error to use a local variable before specifying its value
- C: Run-time error to use a local variable before specifying its value

Compound Statement:
- Java: Declarations statements can be placed anywhere within compound statement
- C: Declaration statements must appear before any other type of statement within compound statement

If Statement
- Java: Controlling *`expr`* must be of type boolean
- C: Controlling *`expr`* must be of some integer type or a pointer (0 => FALSE, non-0 => TRUE)

While Statement
- Java: Controlling *`expr`* must be of type boolean
- C: Controlling *`expr`* must be of some integer type or a pointer (0 => FALSE, non-0 => TRUE)

DoWhile Statement
- Java: Controlling *`expr`* must be of type boolean
- C: Controlling *`expr`* must be of some integer type or a pointer (0 => FALSE, non-0 => TRUE)

For Statement
- Java: Controlling *`expr`* must be of type boolean
- C: Controlling *`expr`* must be of some integer type or a pointer (0 => FALSE, non-0 => TRUE)
- Java: Can declare loop control variable in *`initexpr`*
- C: Cannot declare loop control variable in *`initexpr`*

Break Statement
- Java: Also has "labeled `break`" statement
- C: Does not have "labeled `break`" statement

Continue Statement
- Java: Also has "labeled `continue`" statement
- C: Does not have "labeled `continue`" statement

Goto Statement
- Java: Not provided
- C: Provided (but don't use it!)

formattedio.c (Page 1 of 3)

```
 1: /*-------------------------------------------------------------------*/
 2: /* formattedio.c                                                     */
 3: /* Author: Bob Dondero                                               */
 4: /*-------------------------------------------------------------------*/
 5:
 6: #include <stdio.h>
 7:
 8: /*-------------------------------------------------------------------*/
 9:
10: /* Read from stdin, and write to stdout, one literal of each
11:    fundamental data type.  Ignore the possibility of bad input.
12:    Return 0. */
13:
14: int main(void)
15: {
16:     int iTypical;
17:     unsigned int uiTypical;
18:     long lTypical;
19:     unsigned long ulTypical;
20:     short sTypical;
21:     unsigned short usTypical;
22:     char cTypical;
23:     unsigned char ucTypical;
24:     double dTypical;
25:     float fTypical;
26:     long double ldTypical;
27:
28:     /*---------------------------------------------------------------*/
29:     /* char                                                          */
30:     /*---------------------------------------------------------------*/
31:
32:     printf("\n");
33:     printf("Enter a constant of type char:\n");
34:     /* Place a space before %c to skip leading whitespace
35:        characters. Do not place a space before %c to read
36:        whitespace characters. */
37:     scanf(" %c", &cTypical);
38:
39:     printf("You entered %c.\n", (int)cTypical);
40:
41:     /*---------------------------------------------------------------*/
42:     /* unsigned char                                                 */
43:     /*---------------------------------------------------------------*/
44:
45:     printf("\n");
46:     printf("Enter a constant of type unsigned char:\n");
47:     scanf(" %c", &ucTypical);
48:
49:     printf("You entered %c.\n", (unsigned int)ucTypical);
50:
51:     /*---------------------------------------------------------------*/
52:     /* short                                                         */
53:     /*---------------------------------------------------------------*/
54:
55:     printf("\n");
56:     printf("Enter a constant of type short:\n");
57:     scanf("%hd", &sTypical);
58:
59:     printf("You entered %hd.\n", sTypical);
60:
61:     /*---------------------------------------------------------------*/
62:     /* unsigned short                                                */
63:     /*---------------------------------------------------------------*/
```

```
64:
65:     printf("\n");
66:     printf("Enter a constant of type unsigned short:\n");
67:     scanf("%hu", &usTypical);
68:
69:     printf("You entered %hu.\n", usTypical);
70:
71:     /*-------------------------------------------------------------*/
72:     /* int                                                         */
73:     /*-------------------------------------------------------------*/
74:
75:     printf("\n");
76:     printf("Enter a constant of type int:\n");
77:     scanf("%d", &iTypical);
78:
79:     printf("You entered %d.\n", iTypical);
80:
81:     /*-------------------------------------------------------------*/
82:     /* unsigned int                                                */
83:     /*-------------------------------------------------------------*/
84:
85:     printf("\n");
86:     printf("Enter a constant of type unsigned int:\n");
87:     scanf("%u", &uiTypical);
88:
89:     printf("You entered %u.\n", uiTypical);
90:
91:     /*-------------------------------------------------------------*/
92:     /* long                                                        */
93:     /*-------------------------------------------------------------*/
94:
95:     printf("\n");
96:     printf("Enter a constant of type long:\n");
97:     scanf("%ld", &lTypical);
98:
99:     printf("You entered %ld.\n", lTypical);
100:
101:    /*-------------------------------------------------------------*/
102:    /* unsigned long                                               */
103:    /*-------------------------------------------------------------*/
104:
105:    printf("\n");
106:    printf("Enter a constant of type unsigned long:\n");
107:    scanf("%lu", &ulTypical);
108:
109:    printf("You entered %lu.\n", ulTypical);
110:
111:    /*-------------------------------------------------------------*/
112:    /* float                                                       */
113:    /*-------------------------------------------------------------*/
114:
115:    printf("\n");
116:    printf("Enter a constant of type float:\n");
117:    scanf("%f", &fTypical);  /* %e or %g work identically.  */
118:
119:    printf("You entered %f.\n", (double)fTypical);
120:    printf("You entered %e.\n", (double)fTypical);
121:    printf("You entered %E.\n", (double)fTypical);
122:    printf("You entered %g.\n", (double)fTypical);
123:    printf("You entered %G.\n", (double)fTypical);
124:
125:    /*-------------------------------------------------------------*/
126:    /* double                                                      */
```

formattedio.c (Page 3 of 3)

```
127:     /*-------------------------------------------------------------------*/
128:
129:     printf("\n");
130:     printf("Enter a constant of type double:\n");
131:     scanf("%lf", &dTypical);  /* %le or %lg work identically.  */
132:
133:     /* Note the assymmetry of the following with scanf().  */
134:     printf("You entered %f.\n", dTypical);
135:     printf("You entered %e.\n", dTypical);
136:     printf("You entered %E.\n", dTypical);
137:     printf("You entered %g.\n", dTypical);
138:     printf("You entered %G.\n", dTypical);
139:
140:     /*-------------------------------------------------------------------*/
141:     /* long double                                                       */
142:     /*-------------------------------------------------------------------*/
143:
144:     printf("\n");
145:     printf("Enter a constant of type long double:\n");
146:     scanf("%Lf", &ldTypical);  /* %Le or %Lg work identically.  */
147:
148:     printf("You entered %Lf.\n", ldTypical);
149:     printf("You entered %Le.\n", ldTypical);
150:     printf("You entered %LE.\n", ldTypical);
151:     printf("You entered %Lg.\n", ldTypical);
152:     printf("You entered %LG.\n", ldTypical);
153:
154:     return 0;
155: }
```