Introduction to C

EECS 2031

Song Wang
wangsong@eecs.yorku.ca
eecs.yorku.ca/~wangsong/

Acknowledgement

- Some of the covered materials are based on previous EECS2031 offerings:
  - Uyen Trang (UT) Nguyen, Pooja Vashisth, Hui Wang, Manos Papagelis

Control Statements

- Conditional:

```
if ( <TEST> ) {
    <STATEMENTS>
} else if ( <TEST> ) {
    <STATEMENTS>
} else {
    <STATEMENTS>
}
```

- Loops:

```
while ( <TEST> ) {
    <STATEMENTS>
}

for ( <INIT> ; <TEST> ; <STEP> )
{
    <STATEMENTS>
}

do {

} while ( <TEST> );
```

Switch statements

- Compares an integer value (only works with integers) against multiple options:

```
switch (month) {
case 1: case 3: case 5: case 7: case 8: case 10: case 12:
    days = 31;
    break;
case 4: case 6: case 9: case 11:
    days = 30;
    break;
case 2:
    days = year % 4 ? 28 : 29;
    break;
default:
    error = 1;
}
```

Detecting equal values with branches

```c
#include <stdio.h>

int main(void) {
   int numYears;

   printf("Enter number years married: ");
   scanf("%d", &numYears);

   if (numYears == 1) {
      printf("Your first year -- great!\n");
   }
   else if (numYears == 10) {
      printf("A whole decade -- impressive.\n");
   }
   else if (numYears == 25) {
      printf("Your silver anniversary -- enjoy.\n");
   }
   else if (numYears == 50) {
      printf("Your golden anniversary -- amazing.\n");
   }
   else {
      printf("Nothing special.\n");
   }

   return 0;
}
```

Detecting ranges using logical operators

Logical operator	Description
a && b	***Logical AND*** (&&): true when both of its operands are true
a || b	***Logical OR*** (||): true when at least one of its two operands are true
!a	***Logical NOT*** (!): true when its one operand is false, and vice-versa.

Boolean Data Type

- There was no boolean data type before C99.
- False is 0; any thing else is true (1 and others).
- Example:

```
int valid;
...
if (valid == 0)   =>  if (!valid)
 { ... }
```

```
int speed;
// get the speed from machine
if ( speed )
  printf( "Car is moving/running" );
else
  printf( "Car is idle/stopped" );
```

Boolean Data Type in C99

```c
#include <stdio.h>
#include <stdbool.h>

int main(void) {
   int waitTime;
   int partySize;
   char day;
   bool isLargeParty;
   bool isWeekend;

   // Get day of reservation
   printf("Day of reservation (T/W/R/F/S/U): ");
   scanf("%c", &day);
   if (day == 'F' || day == 'S' || day == 'U') {
      isWeekend = true;
   }
   else {
      isWeekend = false;
   }
   // Get party size
   printf("Enter party size: ");
   scanf("%d", &partySize);
   isLargeParty = (partySize > 6);

   // Determine wait time based on day of week and party size
   if (isWeekend && !isLargeParty) {
      waitTime = 30;
   }
   else if (!isWeekend && !isLargeParty) {
      waitTime = 10;
   }
   else if (isWeekend && isLargeParty) {
      waitTime = 45;
   }
   else {
      waitTime = 15;
   }

   printf("Restaurant wait time is %d minutes.\n", waitTime);

   return 0;
}
```

Order of evaluation

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In `(a * (b + c)) - d`, the + is evaluated first, then *, then -.
!	! (logical NOT) is next	`! x || y` is evaluated as `(!x) || y`
* / % + -	Arithmetic operators (using their precedence rules; see earlier section)	`z - 45 * y < 53` evaluates * first, then -, then <.
< <= > >=	Relational operators	`x < 2 || x >= 10` is evaluated as `(x < 2) || (x >= 10)` because < and >= have precedence over ||.
== !=	Equality and inequality operators	`x == 0 && x >= 10` is evaluated as `(x == 0) && (x >= 10)` because < and >= have precedence over &&. == and != have the same precedence and are evaluated left to right.
&&	Logical AND	`x == 5 || y == 10 && z != 10` is evaluated as `(x == 5) || ((y == 10) && (z != 10))` because && has precedence over ||.
||	Logical OR	|| has the lowest precedence of the listed arithmetic, logical, and relational operators.

Switch statements

```
switch (userChar) {
   case 'A':
      encodedVal = 1;
      break;

   case 'B':
      encodedVal = 2;
      break;

   case 'C':

   case 'D':
      encodedVal = 4;
      break;

   case 'E':
      encodedVal = 5;

   case 'F':
      encodedVal = 6;
      break;

   default:
      encodedVal = -1;
      break;
}
```

String comparisons

Relation	Returns	Expression to detect
str1 less than str2	Negative number	strcmp(str1, str2) < 0
str1 equal to str2	0	strcmp(str1, str2) == 0
str1 greater than str2	Positive number	strcmp(str1, str2) > 0

- Must `#include <string.h>`

- Let `k` be the index of the first character where `str1` and `str2` are different.
  - If `str1[k] < str2[k]` then str1 is “smaller” than str2 (returns a negative number)

- `str1 == str2` is a different comparison (to learn later).

String access operations

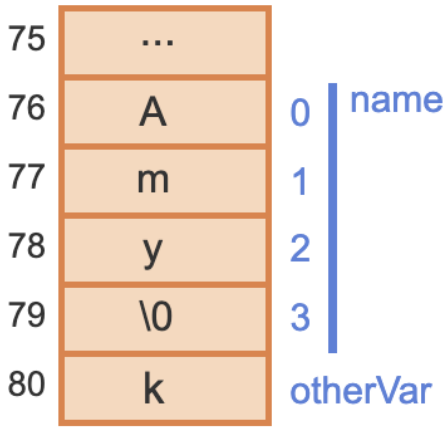
```c
#include <stdio.h>

int main(void) {
    char userWord[6];

    printf("Enter a 5-letter word: ");
    scanf("%s", userWord);

    printf("Scrambled: ");
    printf("%c", userWord[3]);
    printf("%c", userWord[1]);
    printf("%c", userWord[4]);
    printf("%c", userWord[0]);
    printf("%c", userWord[2]);
    printf("\n");

    return 0;
}
```



strlen(): 3
4th element for \0

Working with the end of a string

```c
#include <stdio.h>
#include <string.h>

int main(void) {
   char userCaption[22]; // 20 user char, +1 for period, +1 for null
char
   int lastIndex;
   char lastChar;

   printf("Enter a caption (20 char max): ");
   scanf("%s", userCaption);

   lastIndex = strlen(userCaption) - 1;
   lastChar  = userCaption[lastIndex];

   if ( (lastChar != '.') && (lastChar != '!') && (lastChar != '?') ) {
      // User's caption lacked ending punctuation, so add a period
      strcat(userCaption, ".");
   }

   printf("New: %s\n", userCaption);

   return 0;
}
```

```
Enter a caption (20 char max): Hello_world
New: Hello_world.

...

Enter a caption (20 char max): Anyone_home?
New: Anyone_home?

...

Enter a caption (20 char max): TGIF!
New: TGIF!

...

Enter a caption (20 char max): Another_day.
New: Another_day.

...

Enter a caption (20 char max):
Life_is_sweet
New: Life_is_sweet.
```

Common Errors

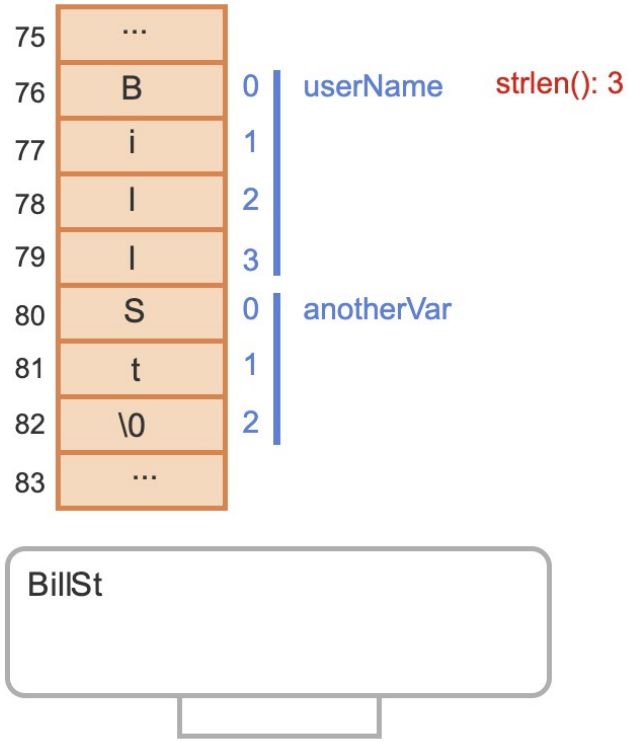
```c
#include <stdio.h>

int main(void) {
 char userName[4] = "Cat";

 userName[0] = 'B';
 userName[1] = 'i';
 userName[2] = 'l';
 userName[3] = 'l';     // Error

 printf("%s\n", userName);

 return 0;
}
```



Character operations

Including the ***ctype.h library*** via `#include <ctype.h>` provides access to several functions for working with characters. ctype stands for character type.

Table 3.15.1: Character functions return values.

isalpha(c)	true if alphabetic: a-z or A-Z	`isalpha('x') // true`
		`isalpha('6') // false`
		`isalpha('!') // false`
				toupper(c)	Uppercase version	`letter = toupper('a') // A`
						`letter = toupper('A') // A`
						`letter = toupper('3') // 3`

isdigit(c)	true if digit: 0-9.	`isdigit('x') // false`
		`isdigit('6') // true`
				tolower(c)	Lowercase version	`letter = tolower('A') // a`
						`letter = tolower('a') // a`
						`letter = tolower('3') // 3`

isspace(c)	true if whitespace.	`isspace(' ') // true`
		`isspace('\n') // true`
		`isspace('x') // false`				

Note: Above, false is zero, and true is non-zero.

See http://www.cplusplus.com/reference/cctype/ for a more complete list (applies to both C and C++).

Example of character operations

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

int main(void) {
   char let0;
   char let1;

   printf("Enter a two-letter state abbreviation: ");
   scanf("%c", &let0);
   scanf("%c", &let1);

   if ( ! (isalpha(let0) && isalpha(let1)) ) {
      printf("Error: Both are not letters.\n");
   }
   else {
      let0 = toupper(let0);
      let1 = toupper(let1);
      printf("Capitalized: %c%c\n", let0, let1);
   }

   return 0;
}
```

```
Enter a two-letter state abbreviation: az
Capitalized: AZ
...
Enter a two-letter state abbreviation: AZ
Capitalized: AZ
...
Enter a two-letter state abbreviation: Mn
Capitalized: MN
...
Enter a two-letter state abbreviation: 5x
Error: Both are not letters.
...
Enter a two-letter state abbreviation: A@
Error: Both are not letters.
```

Conditional expressions

```
if (condition) {
  myVar = expr1;
}
else {
  myVar = expr2;
}
```

```
myVar = (condition) ? expr1 : expr2;
```

Conditional Expressions: Advantage

- Succinct code

- Example 1:

```
for (i = 0; i < n; i++)
  printf( "%6d%c", a[i],
  ( i%10==9 || i==n-1) ? '\n' : ’ ’ );
```

- Example 2:

```
printf( "You have %d item%s.\n", n,
          n==1 ? "" : "s");
```

Floating-point comparison

Start 2x speed

```
numMeters = 0.7;
numMeters = numMeters - 0.4;
numMeters = numMeters - 0.3;

// numMeters expected to be 0,
// but is actually 0.00000000000000000555112

 if (fabs(numMeters - 0.0) < 0.001) {
    // Equals 0.
 }
 else {
    // Does not equal 0.
 }
```

Expected	Actual
0.7	0.69999999999999999555910790
0.4	0.40000000000000000222044605
0.3	0.29999999999999999888977697

0	-0.00000000000000000555111512	numMeters

```
if (numMeters == 0.0) {
   // Equals 0.
}
else {
   // Does not equal 0.
}
```

Bug

Example of floating-point comparison

```c
#include <stdio.h>
#include <math.h>

int main(void) {
   double bodyTemp;

   printf("Enter body temperature in Fahrenheit: ");
   scanf("%lf", &bodyTemp);

   if (fabs(bodyTemp - 98.6) < 0.0001) {
      printf("Temperature is exactly normal.\n");
   }
   else if (bodyTemp > 98.6) {
      printf("Temperature is above normal.\n");
   }
   else {
      printf("Temperature is below normal.\n");
   }

   return 0;
}
```

```
Enter body temperature in Fahrenheit: 98.6
Temperature is exactly normal.

...

Enter body temperature in Fahrenheit: 90
Temperature is below normal.

...

Enter body temperature in Fahrenheit: 99
Temperature is above normal.
```

Floating point: format sub-specifier

```c
#include <stdio.h>

int main(void) {
   double sampleValue1 = 0.2;
   double sampleValue2 = 0.3;
   double sampleValue3 = 0.7;
   double sampleValue4 = 0.0;
   double sampleValue5 = 0.25;

   printf("samplevalue1 using just %%lf: %lf\n",
          sampleValue1);

   printf("sampleValue1 is %.25lf\n", sampleValue1);
   printf("sampleValue2 is %.25lf\n", sampleValue2);
   printf("sampleValue3 is %.25lf\n", sampleValue3);
   printf("sampleValue4 is %.25lf\n", sampleValue4);
   printf("sampleValue5 is %.25lf\n", sampleValue5);

   return 0;
}
```

```
samplevalue1 using just %lf: 0.200000
sampleValue1 is 0.2000000000000000111022302
sampleValue2 is 0.2999999999999999888977698
sampleValue3 is 0.6999999999999999555910790
sampleValue4 is 0.0000000000000000000000000
sampleValue5 is 0.2500000000000000000000000
```

Short circuit evaluation

Operator	Example	Short circuit evaluation
`operand1 && operand2`	**`true`** `&& operand2`	If the first operand evaluates to true, operand2 is evaluated.
	`false` `&& operand2`	If the first operand evaluates to false, the result of the AND operation is always false, so operand2 is not evaluated.
`operand1 || operand2`	**`true`** `|| operand2`	If the first operand evaluates to true, the result of the OR operation is always true, so operand2 is not evaluated.
	`false` `|| operand2`	If the first operand evaluates to false, operand2 is evaluated.

while loops

```c
#include <stdio.h>

int main(void) {
   int currPower;
   char userChar;

   currPower = 2;
   userChar = 'y';

   while (userChar == 'y') {
      printf("%d\n", currPower);
      currPower = currPower * 2;
      scanf("%c", &userChar);
   }

   printf("Done\n");

   return 0;
}
```

Input

```
y y n
```

Output

```
2
4
8
Done
```

for loops

```c
#include <stdio.h>

// Outputs average of list of integers
// First value indicates list size
// Ex: 4  10 1 6 3  yields (10 + 1 + 6 + 3) / 4, or 5

int main(void) {
   int currValue;
   int valuesSum;
   int numValues;
   int i;

   scanf("%d", &numValues); // Gets number of values in list

   valuesSum = 0;

   for (i = 0; i < numValues; ++i) {
      scanf("%d", &currValue); // Gets next value in list
      valuesSum += currValue;
   }

   printf("Average: %d\n", (valuesSum / numValues));

   return 0;
}
```

```
4 10 1 6 3
Average: 5

...

5 -75 -50 30 60 80
Average: 9
```

Increment and Decrement Operators

- ++ or --
- Placing in front: incrementing or decrementing occurs **BEFORE** value assigned

i = 2 and k = 1

```
k = ++i;        i = i + 1; 3
                k = i;     3

k = --i;        i = i - 1; 1
                k = i;     1
```

- Placing after variable: occurs **AFTER** value assigned

i = 2 and k = 1

```
k = i++;        k = i;     2
                i = i + 1; 3

k = i--;        k = i;     2
                i = i - 1; 1
```

Nested loops

```c
#include <stdio.h>

/* Output all two-letter .com Internet domain names */

int main(void) {
   char letter1;
   char letter2;

   printf("Two-letter domain names:\n");

   letter1 = 'a';
   while (letter1 <= 'z') {
      letter2 = 'a';
      while (letter2 <= 'z') {
         printf("%c%c.com\n", letter1, letter2);
         ++letter2;
      }
      ++letter1;
   }

   return 0;
}
```

```
Two-letter domain names:
aa.com
ab.com
ac.com
ad.com
ae.com
af.com
ag.com
ah.com
ai.com
aj.com
ak.com
al.com
am.com
an.com
ao.com
ap.com
aq.com
ar.com
as.com
at.com
au.com
av.com
aw.com
ax.com
ay.com
az.com
ba.com
bb.com
bc.com
bd.com
be.com

...
```

do-while loops

```c
#include <stdio.h>

int main(void) {
  char fill;

  fill = '*';

  do {
     printf("%c%c%c\n", fill, fill, fill);
     printf("%c%c%c\n", fill, fill, fill);
     printf("%c%c%c\n", fill, fill, fill);

     printf("Enter char (q to quit): ");
     scanf(" %c", &fill);
     printf("\n");
  } while (fill != 'q');

  return 0;
}
```

```
***
***
***
Enter char (q to quit):  x

xxx
xxx
xxx

Enter char (q to quit): q
```

break and continue

- A `break` statement in a loop causes an immediate exit of the loop.
- A `continue` statement in a loop causes an immediate jump to the loop condition check.
- `break` and `continue` can sometimes improve the readability of a loop.
- `break` and `continue` should be used only when necessary.

break

```
...
for (i = 0; i < n; i++)
  if (a[i] < 0) /* 1st negative element */
          break;
if (i < n)
  return i;
...
```

continue

```
for (i = 0; i < n; i++) {
  if (a[i] < 0)
        continue;
  a[i]++;
}
```

Variable name scope

```c
#include <stdio.h>

int main(void) {
    // int val1 = userNum;     // ERROR
   int userNum = 2;           // Name valid to main's "}"
   int newNum  = userNum + 1;
   int i;

    for (i = 0; i < newNum; ++i) {
      int valSquared;         // Name valid to for's "}"
      valSquared = userNum * userNum;
      printf("%d squared: %d\n", i, valSquared);
   }

    // printf("Last value: %d\n", valSquared); // ERROR

    return 0;
}
```

for loop index

for loop	Equivalent while loop

`for (int i = 0; i < 5; ++i) {`
`  x = x + i;`
`}`
 	
`x = x + i;  // ERROR`
	`{`
	`  int i = 0;`
	`  while (i < 5) {`
	`  x = x + i;`
	`  ++i;`
	`  }`
	`}`
	`x = x + i;  // ERROR`

Common error: variable re-initialized at each iteration

```c
#include <stdio.h>

int main(void) {
    int i = 0;

    while (i < 5) {
        int tmpSum = 0;
        tmpSum = tmpSum + i; // Logic error: Sum is always just i
        printf("tmpSum: %d\n", tmpSum);
        i = i + 1;
    }

    return 0;
}
```

```
tmpSum: 0
tmpSum: 1
tmpSum: 2
tmpSum: 3
tmpSum: 4
```

Enumerations

```c
#include <stdio.h>

/* Manual controller for traffic light */
int main(void) {
   enum LightState {LS_RED, LS_GREEN, LS_YELLOW, LS_DONE};
   enum LightState lightVal;
   char userCmd;

   lightVal = LS_RED;
   userCmd = '-';

   printf("User commands: n (next), r (red), q (quit).\n\n");

   lightVal = LS_RED;
   while (lightVal != LS_DONE) {

      if (lightVal == LS_GREEN) {
         printf("Green light  ");
         scanf(" %c", &userCmd);
         if (userCmd == 'n') { // Next
            lightVal = LS_YELLOW;
         }
      }
      else if (lightVal == LS_YELLOW) {
         printf("Yellow light  ");
         scanf(" %c", &userCmd);
         if (userCmd == 'n') { // Next
            lightVal = LS_RED;
         }
      }
      else if (lightVal == LS_RED) {
         printf("Red light  ");
```

```
User commands: n (next), r (red), q (quit).

Red light  n
Green light  n
Yellow light  n
Red light  n
Green light  r
Red light  n
Green light  n
Yellow light  n
Red light  q
Quit program.
```

Enumeration Examples

```
enum boolean { NO, YES };
```

- The first name in an enum has value 0, the next 1, and so on, unless explicit values are specified.

```
enum colours { black, white, red, blue, green };

enum escapes { BELL = '\a', BACKSPACE = '\b', TAB = '\t', NEWLINE = '\n', VTAB
  = '\v', RETURN = '\r' };
```

- If not all values are specified, unspecified values continue the progression from the last specified value.

```
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV,
  DEC };

  /* FEB = 2, MAR = 3, etc. */

enum TvChannels {TC_CBS = 2, TC_NBC = 5, TC_ABC = 7};
```