Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CS Basics
10) C Control Structures

E. Benoist & C. Grothoff

Fall Term 2018-19

Computer Science Division

Control Structures for C

- Data Input and Output
  - Single char Input and Output
  - Writing output data
- Control Statements
  - Branching
  - Looping
  - More branching

Data Input and Output

Functions for Data Input and Output

- `getchar()`
  - Read one character
- `putchar()`
  - Write one character
- `scanf()`
  - Read: integers, strings, floating point numbers, hexadecimal, ...
- `printf()`
  - Write: Integers (decimal / hexadecimal / octal), strings, floating point numbers, characters, ...
- `gets()`
  - Read a string
- `puts()`
  - Write a string

Outline of a typical C program

```c
#include <stdio.h>
main(){
   char c,d;     /* declarations */
   float x,y;
   int   i,j,k;
    ...
   i=getchar();          /* character input  */
   scanf("%f", &x);      /* floating point input */
   ...
   putchar(d);           /* character output*/
   printf("%3d␣%7.4f",k, y);
                         /* numerical output */
}
```

Single character Input and Out

Single character Input

- **Function** `int getchar(void)`
  - Part of the standard I/O library
  - Returns one character from a standard input (stdin)
- **Syntax**
  - `int c = getchar();`
- **Example**

```
int i;
char c;

i = getchar();
if (EOF != i)
  c = (char) i;
```

- **End of file**
  - character `EOF` is returned at end of file
  - `EOF` is a constant defined in `stdio.h`

Single character output

- **The `int putchar(int)` function**
  - Writes on stdout the character given as argument
- **Example**

```
char c;
c='a';
putchar(c);
```

Entering input data

- **The `scanf()` function**
  - Reads input from standard input
  - Parse inputs in variables
- **Syntax**

```
scanf(control string, arg1, arg2, ... , ↘
→argn
```

  - Arguments represent pointers (i.e. addresses) where the values can be stored.
- **Example**

```
char str1[20];
int i1;
float f1;
...
scanf("%19s␣%d␣%f", str1,&i1,&f1);
```

- Arrays are addresses (address of the first element)

Formats for scanf

- **Integers**
  - %d integers
  - %ld long int
  - %hd short int
- **Unsigned integers**
  - %u integer
  - %hu unsigned short
  - %lu unsigned long int
  - %x hexadecimal
  - %X hexadecimal (capitals)
- **Floats**
  - %f float
  - %lf double
  - %Lf long double

Scanf Example

- **Scan three numbers**

```
/* Example 1 Scanf */
int i1;
double d1;
char c1;
puts("type␣an␣integer,␣a␣float␣and␣a␣char");
scanf("␣%d␣%lf␣%c",&i1,&d1,&c1);
printf("␣i1=%d,␣d1=%lf␣c1=%c\n",i1,d1,c1);
/*
   if input is: 1234 10.5 T
   Output is i1=1234, d1=10.500000 c1=T
*/
```

Using scanf with format

- **Define the length to be parsed**
  - One can set for each item, the number of chars parsed
- **Example**

```
int i2;
double d2;
char c2;
puts("type␣an␣integer␣(3␣digits),␣a␣float␣↘
→(5␣chars)␣and␣a␣char");
scanf("␣%3d␣%5lf␣%c",&i2,&d2,&c2);
printf("␣i2=%d,␣d2=%lf␣c2=%c\n",i2,d2,c2);
/* if input is: 1234 10.5 T
Output is: i2=123, d2=4.000000 c2=1
```

sscanf() (String Scan Format)

- **Similar to** `scanf()`
  - Scan not from stdin, but rather from a given string
  - String can be read using `gets()`
- **Example**

```
char line[256];
int i=0;
int a,b,c,d;
while(gets(line)){
   sscanf(line,"%d␣%d␣%d␣%d",&a,&b,&c,&d);
   printf("%d␣+␣%d␣+␣%d␣+␣%d␣=␣%d\n",a,b,c,↘
   →d,(a+b+c+d));
}
```

Writing output data

The printf() function

- **Format an output using a string**
  - syntax

    ```
    printf(control string, arg1, arg2, arg3↘
    →,... , argn)
    ```

  - Each arg correspond to one expression in control string
  - an expression starts with %
- **Possible conversion characters**
  - `c` single character
  - `d` signed decimal integer
  - `e` floating-point number with an exponent
  - `f` floating-point number without exponent
  - `g` use e or f type depending on value
  - `i` signed decimal integer
  - `o` octal integer
  - `s` string
  - `u` unsigned decimal integer
  - `x` hexadecimal integer

Field width and precision specification

- **One can state the size of the generated string**
  - Two numbers:
    minimal length of generated string "." precision
- **Minimal length**
  - If the content is not long enough, the rest is filed with spaces
- **Precision**
  - For strings: Maximum number of characters
  - For floating point numbers: number of decimal places

Example

- **length . precision**

```
/* Example 1 Printf */
double d1 = 101.593039;
printf("%f␣%.2f␣%.0f␣%2.2f␣\n",d1,d1,d1,d1↘
→);

/* Output: 101.593039 101.59 102 101.59
 */

/* Example 2 Printf */
char line[]="ABCDEFGHIJKLMNOP";
printf("%10s␣%15s␣%15.5s␣%.10s\n",line,↘
→line,line,line);
/* Output:
   ABCDEFGHIJKLMNOP ABCDEFGHIJKLMNOP ↘
   →              ABCDE ABCDEFGHIJ
*/
```

printf() Flags

- **Each character group can include a flag**
  - – left justified
  - + a sign will precede each numerical data item.
  - 0 zeros appear instead of blanks
  - ’ ’ (blank space) a blank space will preced each positive number.
  - # (with o- and x-type) octal and hexa numbers are preceded by 0 and 0x, respectively
  - # (with e-, f- and g-type) Decimal point must be present even if not required
    for g it prevents any truncation of zeros.

Flags: Example

- **Flags change the way expressions are printed**

```
int i1 = 1234;
double x = 12.0, y = -3.3;

printf(":%6d␣%7.0f␣%10.1e:\n",i1,x,y);
printf(":%-6d␣%-7.0f␣%-10.1e:\n",i1,x,y);
printf(":%+6d␣%+7.0f␣%+10.1e:\n",i1,x,y);
printf(":%-+6d␣%-+7.0f␣%-+10.1e:\n",i1,x,y↘
→);
printf(":%7.0f␣%#7.0f␣%7g␣%#7g:\n",x,x,y,y↘
→);
/* Output:
   :  1234      12   -3.3e+00:
   :1234   12      -3.3e+00  :
   : +1234     +12   -3.3e+00:
   :+1234  +12      -3.3e+00  :
   :      12      12.     -3.3 -3.30000:
*/
```

Functions: gets() and puts()

- **Read a string from stdin:** `gets()`
  - Get a String
  - Populate a string given as argument
- **Write a string to stdout:** `puts()`
  - Put a String
  - Write the string given as input to standard output
- **Example**

```
#include <stdio.h>
main(){
   char line[80];
   puts("Hello, what is your name?");
   gets(line);
   printf("Hello ");
   puts(line);
}
```

Typical interactive program

```
#include <stdio.h>
main(){
   char name[80];
   int nbMarks,i;
   float mark, sum, avg;
   puts("Hello, what is your name?");
   /* scan a line: everything not \n */
   scanf(" %[^\n]",name);
   printf("Number of marks?");
   scanf(" %d",&nbMarks);
   for(i=0;i<nbMarks;i++){
      printf("Enter Mark %d:",(i+1));
      scanf(" %f",&mark);
      sum += mark;
   }
   avg = sum / nbMarks;
   printf("%s, your average mark is: %.1f \n",↘
   →name,avg);
}
```

Control Statements

Branching

Branching: IF, ELSE

- **Syntax**
  - Similar to Java

```
if (expression) statement 1 else ↘
→statement 2
```

  - If `expression` is true, `statement 1` is executed, (and not `statement 2`)
  - If `expression` is false, `statement 2` is executed (and not `statement 1`)
- **Examples**

```
if (status == 'S')
    tax = 0.20 * pay;
else
    tax = 0.14 * pay;
```

Branching (Cont.)

- **Block statements can contain more than one instruction**

```
if(circle){
    scanf("%f",&radius);
    area = PI * radius * radius;
    printf("Area␣of␣the␣circle␣=␣%f␣\n",area)↘
    →;
}
else {
    scanf("%f␣%f", &length, &width);
    area = length * width;
    printf("Area␣of␣rectangle␣=␣%f",area);
}
```

Looping

Looping: the WHILE statement

- **Syntax**
  - Similar to Java

    ```
    while (expression) statement
    ```

  - The `statement` will be executed as long as the `expression` is true
- **Example**

```
while (digit <=9) {
    printf("%d\n", digit);
    ++digit;
}
```

WHILE (Cont.)

- **Example: compute average of marks**

```
#include <stdio.h>
main(){
   char name[80];
   int nbMarks,i=0;
   float mark, sum, avg;
   puts("Hello, what is your name?");
   /* scan a line: everything not \n */
   scanf(" %[^\n]",name);
   printf("Number of marks?");
   scanf(" %d",&nbMarks);
   while (i<nbMarks){
      i++;
      printf("Enter Mark %d:",(i));
      scanf(" %f",&mark);
      sum += mark;
   }
   avg = sum / nbMarks;
   printf("%s, your average mark is: %.1f \n",name,↘
   →avg);
}
```

More Looping: DO ... WHILE

- **Syntax**
  - ```
    do statement while(expression)↘
    →;
    ```
  - Executes statement as long as expression is true
  - The statement is executed at least once, since expression is evaluated afterwards.
- **Example**

```
do
   printf("%d\n",digit++);
while(digit <= 9);
```
```

DO ... WHILE (Cont.)

- **Transform a string to uppercase**[1]

```
char line[80];
int i = 0, size=0;
/* Scan one line (until a \n is read) */
puts("Type␣a␣line␣(to␣be␣transformed␣in␣↘
→uppercase):");
scanf("%[^\n]",line);
do
  ++size;
while(line[size]!=0);
do {
  putchar(toupper((unsigned char) line[i])↘
  →);
  i++;
} while (i<size);
putchar('\n');
```

[1] Need the library <ctype.h>

Loops with FOR

- **Syntax**
  - for (expression 1; expression 2; expression 3) statement
  - expression 1: is used to initialize some parameter
  - expression 2: represents the condition that must be true for the loop to continue execution
  - expression 3: used to alter the value of the parameter
- **Example**

```
for(digit = 0; digit <=9; ++digit){
   printf("%d\n",digit);
}
```

FOR (cont.)

- **Visit a string**

```
for(i=0;i<size;i++){
   putchar(toupper((unsigned char) line[i])↘
   →);
}
putchar('\n');
```

- **Compute an average**

```
for(i=0;i<nbMarks;i++){
   printf("Enter␣Mark␣%d:",(i+1));
   scanf("␣%f",&mark);
   sum += mark;
}
avg = sum / nbMarks;
```

More branching

The SWITCH statement

- **Cause a particular group of statements to be chosen from several available groups**
  - Similar to Java
- **Syntax**

```
switch (expression) {
  case expression 1:
        statement 1
        statement 2
   case expression 2:
        statement 3
    ....
    case expression n:
        statement m-1
    default:
        statement m
}
```

SWITCH (Cont.)

- **Syntax**
  - `expression` is evaluated
  - `expression 1` to `expression n` must represent constant, integer-valued expressions: either an integer constant or a character constant.
  - control is transfered, to the first expression that matches `expression`
  - if the statement does not contain a `break`, then the statements following the matched expression are executed.
- **Default case**
  - The case `default` matches any value of `expression`.

SWITCH: example

- **Ask for a letter**

```
puts("Enter one letter for a color (RGB):");
int choice;
choice = getchar();
switch(choice){
case 'r':
case 'R':
   printf("Red");
   break;
case 'g':
case 'G':
   printf("Green");
   break;
case 'b':
case 'B':
   printf("Blue");
   break;
default:
   printf("Not a valid color");
}
```

The BREAK statement

- **Is used to terminate a loop or exit from a SWITCH**
  - Can be used within `for`, `while`, `do - while` or `switch`
- **Syntax**

```
break;
```

- **Example (switch)**

```
case 'B':
   printf("Blue");
   break;
```

BREAK (Cont.)

- **Example with `while`**

```
int val, exponent;
int i=0;
int res = 1;
puts("type␣value␣and␣exponent:");
scanf("␣%d␣%d",&val,&exponent);

while(1){
  if(i++ == exponent){
    break;
  }
  res *= val;
}
printf("%d␣power␣%d␣=␣%d\n",val,exponent,↘
→res);
```

The CONTINUE statement

- **Is used to bypass the remainder of the current pass throug a loop**
  - The loop does not terminate
  - The remaining loop statements are skipped and the computation proceeds to the next pass through the loop
  - BREAK: go out of loop
  - CONTINUE: go to the next pass of the loop
- **Example**

```
for(i=0; i< 10; i++){
   puts("Type␣a␣number:");
   scanf("%d",&val);
   if(val<0){ /* A negative number does not↘
   → have a sqrt*/
      continue;
   }
   double res = sqrt(val*1.0);
   printf("Square␣root␣of␣%d␣is␣%lg\n",val,↘
   →res);
```

The GOTO statement

- **Very simplar to JMP in Assembly**
  - Go from one place to a label
  - Label must be in the same function (we see functions next week)
  - The label must be unique inside the function
- **Syntax**
  - `goto label`
  - and somewhere else in function
    `label: statement`

Use of `goto` in production code requires a license:

- You must have a Bachelor before using `goto` in C
- You must have a Master before using unstructured `goto`
- You must have a PhD before using `longjmp()`

Example: GOTO

- **Goto error message if an error occures**

```
for(i=0; i< 10; i++){
    puts("Type a number:");
    scanf("%d",&val);
    if(val<0){ /* A negative number does not↘
    → have a sqrt*/
      goto errorNegativeNumber;
    }
    double res = sqrt(val*1.0);
    printf("Square root of %d is %lg\n",val,↘
    →res);

 }
 return;
errorNegativeNumber:{
    printf("Negative number do not have any ↘
    →root");
 }
```

Conclusion

- **Input and Output**
  - Read a string, parse it
  - Print out formated output
  - More about files and other functions later
- **Control statement**
  - Similar to Java
  - Some controls are “low level”
    `SWITCH` (with integers)
    `GOTO` (similar to assembly language)

Bibliography

- This course corresponds to chapters *4*, (*5*) and *6* of the course book:
  Schaum's OuTlines, Programming with C (second edition), *Byron Gottfied*, Mc Graw-Hill, 1996