CSD101: Introduction to computing and programming (ICP)

Conditional construct switch

The `switch` construct allows us to switch between different alternatives based on the value of an expression. Execution starts at the matching `case` and continues below. Cases and default can occur in any order. An explicit `break` statement is needed to avoid cascading cases and exiting the switch statement.

```
switch (<expr>) {
    case <const-expr>: <statement>
    case <const-expr>: <statement>
    ...
    default: <statement>
}
```

printf structure

- The `printf` function call structure is below.
  `printf("<format spec.>",<arguments>)`
- The `<format spec.>` contains 0 or more conversion specifiers signalled by the % sign. For each conversion specifier there should be an argument in `<arguments>` of the corresponding type.
- Till now the conversion specifiers have been simple indicators of the type. For example, `%d` for `int` , `%f` for `float` etc.
- The conversion specifier can have a much more complex structure with optional elements that allow one to control the formatting. This is discussed in the following slides.

Format specification of `printf` I

The structure of the conversion specifier in the format specification is given below.

%	[<flags>]	[<Min. width>]	[<precision>]	[<size>]	<conv. char>

- Only % and <conv. char> are necessary. Others are optional.
- <Min. width> gives the minimum width in characters of the entire field. For a string (%s) it is right justifed.
- The <flags> are given below.

Flag	**Meaning**
−	Left justify
+	Print +/− sign of numeric value
space	Print space if no sign
0	Pad with leading 0s
#	Special print spec. for float, octal, hex

Format specification of `printf` II

The behaviour for the # flag is given below.

Flag	Meaning
%#0	Adds leading 0 to octal number
%#*x* or *X*	Adds leading 0*x* or 0*X* to hex
%#*f* or *e*	Always prints decimal point
%#*g* or *G*	Prints trailing 0s and decimal point

- `<precision>` gives the number of digits after the decimal point for `float` values. If present it should be preceded by a decimal point. For string values printed with %s only that many characters will be printed
- The `<size>` modifier behaviour is given below.

Flag	Conv. char	Meaning
l	d,i,o,u,x	`long int`
h	d,i,o,u,x	`short int`
l	e,f	`double`
L	e,f	`long double`

Format specification of `printf` III

- `printf` conversion char description[1]

`d,i`	`int`; signed decimal notation.
`o`	`int`; unsigned octal notation (without a leading zero).
`x,X`	`unsigned int`; unsigned hexadecimal notation (without a leading `0x` or `0X`), using `abcdef` for `0x` or `ABCDEF` for `0X`.
`u`	`int`; unsigned decimal notation.
`c`	`int`; single character, after conversion to `unsigned char`
`s`	`char *`; characters from the string are printed until a `'\0'` is reached or until the number of characters indicated by the precision have been printed.
`f`	`double`; decimal notation of the form `[-]`*mmm.ddd*, where the number of *d*'s is given by the precision. The default precision is 6; a precision of 0 suppresses the decimal point.
`e,E`	`double`; decimal notation of the form `[-]`*m.ddddd*`e+/-`*xx* or `[-]`*m.ddddd*`E+/-`*xx*, where the number of *d*'s is specified by the precision. The default precision is 6; a precision of 0 suppresses the decimal point.
`g,G`	`double`; `%e` or `%E` is used if the exponent is less than -4 or greater than or equal to the precision; otherwise `%f` is used. Trailing zeros and a trailing decimal point are not printed.
`p`	`void *`; print as a pointer (implementation-dependent representation).
`n`	`int *`; the number of characters written so far by this call to `printf` is *written into* the argument. No argument is converted.
`%`	no argument is converted; print a %

[1] From Kernighan, Ritchie

scanf

- `scanf` is similar to printf in structure. It has a format specification and arguments.
- The format specification contains the conversion specifications and reads in values based on the conversion specification till it hits a white space character (blank, tab, newline,carriage return, vert tab, formfeed). The argument corresponding to each conversion specification must be an address of a variable of a compatible type.
- Any characters specified in the format specification other than the conversion specification must match exactly in the input.
- scanf returns the number of items converted and assigned or an error (or EOF - end of file) if there is a converion error (end of input reached before conversion).

scanf - conversion table[2]

Table B.2 *Scanf Conversions*

Character	Input Data; Argument type
`d`	decimal integer; `int*`
`i`	integer; `int*`. The integer may be in octal (leading `0`) or hexadecimal (leading `0x` or `0X`).
`o`	octal integer (with or without leading zero); `int *`.
`u`	unsigned decimal integer; `unsigned int *`.
`x`	hexadecimal integer (with or without leading `0x` or `0X`); `int*`.
`c`	characters; `char*`. The next input characters are placed in the indicated array, up to the number given by the width field; the default is 1. No `'\0'` is added. The normal skip over white space characters is suppressed in this case; to read the next non-white space character, use `%1s`.
`s`	string of non-white space characters (not quoted); `char *`, pointing to an array of characters large enough to hold the string and a terminating `'\0'` that will be added.
`e,f,g`	floating-point number; `float *`. The input format for `float`'s is an optional sign, a string of numbers possibly containing a decimal point, and an optional exponent field containing an `E` or `e` followed by a possibly signed integer.

[2] From Kernighan, Ritchie

Other input-output functions

- The standard **C** library also provides single character input-output.
- `getchar()` reads a single character from the standard input stream (keyboard) and returns an `int` value corresponding to the character read. If the end of the input stream is reached it returns the `EOF` character (end-of-file). `EOF` is returned when there is no more input and the end of the input stream is reached.
- `putchar(ch)` writes the character ch on the standard output (terminal/screen) and returns an integer corresponding to the character ch.
- Note that **C** encodes characters as unsigned integers so one can do arithmetic operations on characters as if they were integers. This is actually a weak point of **C** .

Block

Definition 3 (Block)

A block is a function or any set of statements enclosed within $\{...\}$ *(curly brackets). Variable declarations can be made within blocks. The* `for` *loop also forms a block.*
C *versions from C99 onwards allow declarations anywhere in the body of a function.*
Variables declared within a block are alive and accessible only within the block. Storage is allocated when a declaration is encountered and de-allocated when the block is exited.

Scope of variables/names

Definition 4 (Scope)

The scope of a variable or name is the body of the program text where it is accessible.

- The scope of a variable or name extends from the point of declaration till the end of the block in which it occurs.
- **Global/ external** variables are those that are declared outside any function.
- Variables declared within a function or a block are called local variables.
- Function names and global or external variables have file scope. That is they are accessible in the file where they have been declared.
- A function **cannot** be declared within a function.