CSD101: Introduction to computing and programming (ICP)

Quick overview of 8 main **C** constructs/ features

The eight main constructs/features of **C** required to write programs:

- Preamble - linking section.
- Declarations.
- Expressions.
- Conditionals.
- Loops.
- Functions.
- Input, output.
- Compound data values - data structures.

Notational conventions

- A word within angular brackets denotes an entity of the kind the word represents. For example: `<var>` stands for any variable name, `<type>` stands for any type, `<init>` stands for any kind of initializing expression, etc.
- When an entity is within square brackets then it is optional. For example, `<type> <var>[=<init>]` means that `=<init>` is optional.
- Other characters/strings shown are required/ necessary. For example in `<type> <var>[=<init>];` the semicolon `;` is required, similarly in the optional `[=<init>]` the `=` is necessary if the optional part is present.
- `...` means the previous unit repeats arbitrarily many times. Of course, in practice there is a finite limit imposed by the compiler.

We will see concrete examples in **C** code.

Preamble or linking section

- Include directives for libraries.
  `#include<<lib-name>>`
  Example:
  `#include<stdio.h>`
  Note that there is no semicolon at the end so these are not statements.
- Define directives for defining constants.
  `#define <var> <const-exp>`
  Example:
  `#define TRUE 1`
  Functions similar to a macro. 1 will be substituted wherever `TRUE` occurs in the code.
  The following will be discussed later in the course.
- Macros.
- Conditional compilation/ inclusion directives.

The preamble is processed by the **C** pre-processor.

Declarations, function defn.

- Two main kinds of declarations.
- Variable declarations.
  `<type> <var1>[=<init1>][, ...];`
  Example: `int i, sum=0;`
- Function declarations.
  `<return-type> <fn-name>([<type1> [<arg1>][, ...]]);`
  Example:
  `int max(int a, int b);` or `int max(int, int);`
- Functions can also be declared and defined simultaneously.
  `<return-type> <fn-name>([<args>]) {<stmt>}` where
  `<args>` is `<type1> <var1>[, ...]`

Expressions

- An expression yields a value. So, they can occur wherever values can legally occur.
- An expression can be defined by the following recursive rules. The symbol `:=` stands for defined as:

```
<expr>   :=   <const>
         :=   <var>
         :=   <uop><expr>
         :=   <expr1><bop><expr2>
         :=   <cond-expr>?<expr-true>:<expr-false>
         :=   <var>=<expr>
```

Legend: `<uop>`: unary operator, `<bop>`: binary operator.
The assingment operator (=) is both an expression and a statement.
`x=y=z*z` is legal, both `x` and `y` will have value `z*z`.

Examples:
a) `10` b) `3.14159` c) `pi=3.14159` d) `-pi` e) `x+y` f) `(i>0)?1:0`

Conditionals

- Conditionals are statements.
- if-then conditional.
  `if (<cond-expr>) <stmt-true>;`
  Example:
  `if (i==10) j=i*i;`
- `if (<cond-expr>) <stmt-true> else <stmt-false>`
  Example:
  `if (i>0) j=i*i; else j=i*i*i;`
- The `switch` statement will be discussed later.

Loops

Three types of loops.

- For loop.
  `for (<var-loop>; <cond-expr>; <inc-expr>) <stmt>`
  Example:
  `for (i=1; i<=n; i=i+1) sum=sum+i;`
- While loop.
  `while (<cond-expr>) <stmt>`
  Example:

```
while (i>0) {
    r=i%10;
    i=i//10;
}
```

- Do-while loop.
  `do <stmt> while <cond-expr>;`
  Example:

```
do
    r=i%10
    i=i//10
while (i>0);
```