CSCI 2132
Software Development

Lecture 8:

Introduction to C

Instructor: Vlado Keselj

Faculty of Computer Science

Dalhousie University

Previous Lecture

- Filename substitution (wildcards)
- Regular expressions
  - basic regular expressions
  - grep, filters

Some Interesting grep Options

- These are some interesting grep options that can be used:
  - `-n`: Output lines preceded by line numbers
  - `-i`: Ignores case
  - `-v`: Output lines that don't match
  - `-w`: Restricts matching to whole words only

Grep Variations

- `grep` : the standard grep
- `grep -F` (or `fgrep`) : searching for fixed strings
- `grep -E` (or `egrep`) : support for extended regular expressions

Extended Regular Expressions (ERE)

- Include matacharacters: `? + | ( ) {`
- These metacharacters can still be used with a backslash; e.g., `\?`
- Back-referencing; e.g., `(...)\1`
- Further extension: PCRE — Perl-Compatible Regular Expressions

Examples of Extended Regular Expressions

- `[0-9]?[0-9][a-z]+`
- `(Mon|Wed|Fri)+`
- `(.)(.).*\2\1`
- `([0-9]{3},){2,5}[0-9]{3}`

C Programming Language

- C is originally invented as a language for writing an operating system and other system software by Denis Ritchie
- C optimizes for machine efficiency at the expense of increased implementation and debugging time
- A central difficulty in C programming: programmers must do their own memory management
- C assumes that you know what you are doing

Writing a Simple Program

- `hello.c` — the first C program from K&R

```
#include <stdio.h>

int main() {
    printf("hello, world\n");
    return 0;
}
```

- We can type this program using *emacs*

Compiling and Running a Simple C Program

- `gcc hello.c` — to compile the program
- `ls -l` — to verify output in `a.out`
- `./a.out` — to run the output
- You can explore Emacs and other tools about how to do this faster

From Source Code to Executable

- Three steps:
  - **Preprocessing** (by a preprocessor): modifies the program by following preprocessor directives
  - **Compiling** (by a compiler): translates modified code into object code (machine instructions)
  - **Linking** (by a linker): combines object code and additional code and produces an executable program
- `gcc` automatically executes these three steps
- Other approach to running programs: interpretation (e.g., shell scripts, Perl, Python)

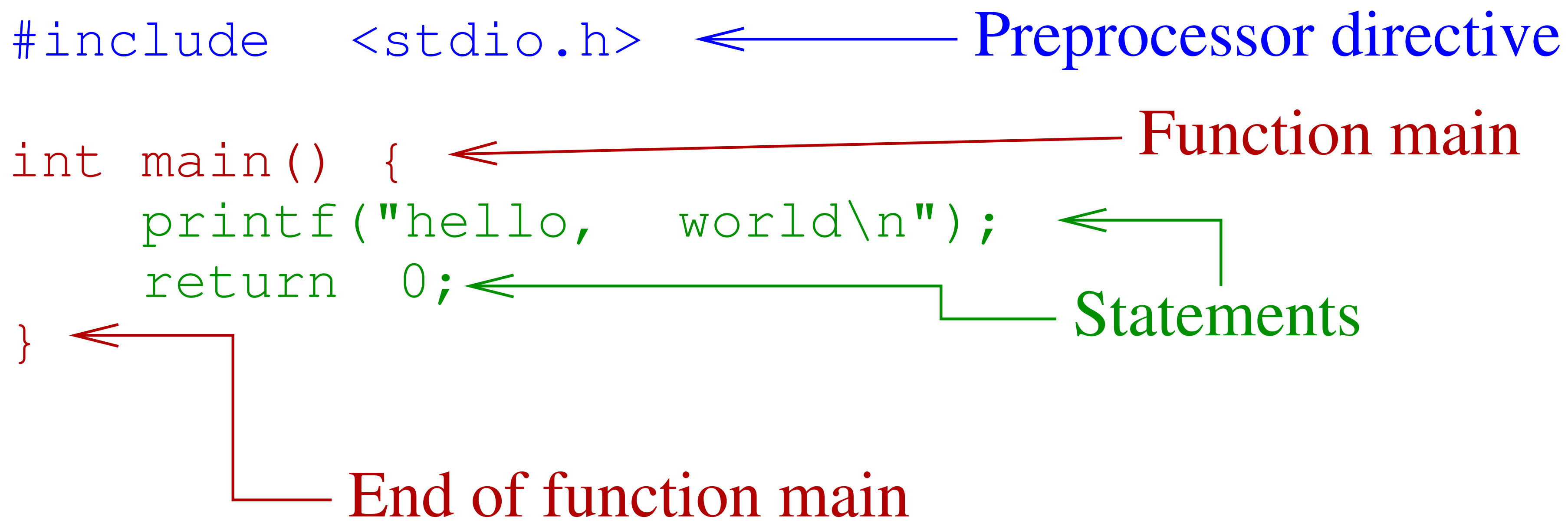
General Form of a Simple Program

```
directives
int main() {
  statements
}
```

or

```
directives
int main(void) {
  statements
}
```

Hello-world Example



```
#include  <stdio.h>

int main() {
    printf("hello,  world\n");
    return  0;
}
```

Functions

- Building blocks from which C programs are constructed
- A function is a group of statements given a name
- **Library functions:** functions provided as a part of the C implementation; e.g., `printf`
- **Main function:** the function that is called automatically when the program is executed
- `int main()` or `int main(void)` means that main returns an integer value, and does not take any parameters
- Nested functions not allowed by standard, but gcc allows them

Statement

- A command to be executed when the program runs
- Must end with a semicolon
- Examples:

```
printf("hello, world\n");
return 0;
```

Printing Strings

- `printf` can print to the standard output a string literal—a series of characters enclosed between `"` and `"`
- Newline character: `\n`
- Examples:

```
printf("hello, ";
printf("world\n");
printf("hello, \nworld\n");
```

- Similar to Java, string literals can include other escape sequences: `\t`, `\r`, `\\`, `\a`, `\b`, `\f`, `\v`, `\'`, `\"`, `\ooo`, `\xHH`, and `\?`.

Comments

- `/*` comments (one or more lines) `*/`
- Example:

```
/* Name: hello.c
   Purpose: prints hello, world
   Authors: K&R
 */
```

- C99 standard: `//` comments (to the end of line)

Variables

- Types
  - Each variable must have a type
- Examples
  - `int` — integers
  - `float` — floating-point numbers
  - `double` — floating-point with double precision
  - `char` — characters
- We will see later how to build more complex types

Declarations

- Variables must be declared before use
- Syntax: *type name*`;`
- Examples:

```
int height;
float profit;
```

- In C89 or earlier, declarations must precede statements in any block of code
- No such restrictions in C99

Operators

- A rich and powerful set of operators was one of the strong novelties of C
- Some operators (in increasing precedence):
  - parentheses `()`
  - unary + and −, ++, −−
  - binary `*`, `/`, `%`
  - binary + and -
  - comparison: <, <= >, >=
  - equality: == and `!=`
  - assignment: =, +=, −=, `*=`, `/=`, `%=`,

Printing Variables

- Printing an integer:

```
printf("Height: %d\h", height);
```

- Printing a floating-point number:
  - printing with a default value of 6 decimal digits:

    ```
    printf("Profit: %f\n", profit);
    ```

  - printing 2 digits after the decimal point:

    ```
    printf("Profit: %.2f\n", profit);
    ```

Initialization

- Variables may have a random value if declared and not initialized
- Declare and initialize in one step:

```
int height = 8;
double profit = 1030.56;
float profit = 1030.56f;
char c = 'A';
char b = '\n';
```

Reading Input: scanf

- Reading an `int` value:
  ```
  scanf("%d", &height);
  ```
- Reading a `float` value:
  ```
  scanf("%f", &profit);
  ```
- Reading a `double` value:
  ```
  scanf("%lf", &precise_profit);
  ```
- Reading an `char` value:
  ```
  scanf("%c", &ch);
  ```

Defining Names for Constants

- Macro definition (preprocessor directive):

```
#define PI 3.14159f
```

- or simply

```
#define PI 3.14159
```

- Preprocessor will replace each occurrence of token `PI` with the number
- A macro definition:
  - does not define a variable
  - is oblivious about the content of the replacement
- Macro replacement can be any sequence of tokens

Example: Expression as a Macro

- The value of a macro can be an expression:
  ```
  #define RECIPROCAL_OF_PI (1.0/3.14159)
  ```
- Important to remember to put parentheses `()` around if using an expression
- Example:
  ```
  double pi = 1.0 / RECIPROCAL_OF_PI;
  ```
- What would happen if we did not have parentheses?
- Convention: uppercase letters are used for constants being defined as macros

Identifiers

- Names for variables, functions, macros, etc.
- May contain letters, digits, and underscores
- Must begin with a letter or underscore
- It is good idea to avoid using underscore as the starting character for now

Example

- Suppose that we write a program for a cashier working in a retail store
- When a customer pays certain amount for a product of certain price, before HST, we want to calculate the balance to be returned to the customer.
- Design:
  - Read price, payment, calculate, print the result
  - HST can be defined as a macro constant, also called symbolic constant

```c
#include <stdio.h>

#define HST 0.15

int main() {
    double price, payment, balance;

    printf("Enter price: ");
    scanf("%lf", &price);

    printf("Enter payment: ");
    scanf("%lf", &payment);

    balance = payment - price * (1.0 + HST);
    printf("Balance to be returned to customer:"
           " %.2f\n", balance);
    return 0;
}
```