Software Design 1
Practical 1

Prepared by:
Anh Duy Nguyen
(251163733)

Prepared for:
SDN150S
Department of Electrical, Electronic and Computer Engineering
Cape Peninsula University of Technology

March 3, 2025

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

D. Nguyen

March 3, 2025

Anh Duy Nguyen

Date

Contents

Chapter 1

Practical 1

1.1 Question 1: Incorrect Radix Conversion

1.1.1 Analysis

The original program fails to produce the expected output due to improper use of format specifiers. The primary issues are as follows:

- `%i` and `%d` are used for signed integers, but `Value` is declared as an `unsigned int`. This can lead to incorrect interpretation of large values.
- `%c` is used to print a character, which is not the intended behavior for numerical conversions.
- `%o` is used for octal output but was incorrectly placed under the 'character' label, making the output misleading.

The value `0xFFFFFFF2` represents `4294967282` in decimal (when treated as an unsigned integer). However, if mistakenly interpreted as a signed integer on a 32-bit system, it may be read as a negative value due to overflow, further contributing to incorrect output.

1.1.2 Correction

To ensure correct output representation, the appropriate format specifiers should be used:

- `%u` is used for printing an unsigned integer in decimal format.
- `%o` correctly prints the octal (base 8) representation of the value.
- `%X` (uppercase) or `%x` (lowercase) correctly prints the hexadecimal (base 16) representation.
- `%c` prints the corresponding ASCII character for the given integer value. This works if the integer falls within the valid ASCII range (0–255), but for large values like `0xFFFFFFF2`, the output is unpredictable.

Additionally, the original program incorrectly assigned `%o` to the 'character' output. Since octal format is unrelated to character representation, this has been corrected in the revised/corrected implementation.

See Next Page for Corrected Code Implementation

1.1.3 Correct C Code Implementation

```c
#include <stdio.h>

int main() {
    unsigned int Value = 0xFFFFFFF2; // hexadecimal representation

    // Corrected print statements
    printf("Decimal: %u\n", Value);
    printf("Octal: %o\n", Value);
    printf("Hexadecimal: %X\n", Value);
    printf("Character: %c\n", Value);

    return 0;
}
```

Listing 1.1: Corrected Radix Conversion Program

If executed, the corrected program will produce the following output (on a typical 32-bit system):

```
Decimal: 4294967282
Octal: 37777777762
Hexadecimal: FFFFFFF2
Character: ?
```

- The decimal output `4294967282` correctly reflects the unsigned integer value.
- The octal output `37777777762` correctly represents the value in base 8.
- The hexadecimal output `FFFFFFF2` correctly prints the number in base 16.
- The character output is unpredictable because `0xFFFFFFF2` exceeds the valid ASCII range. Most systems will display a placeholder or an unrecognizable symbol. One could also delete the character output code for errorless code if necessary.

1.2 Question 2: Calendar Program

1.2.1 Errors in the Original C Program

The original program aimed to calculate the number of days in different parts of the year but contained several critical errors as summarised below:

1. **Invalid variable names:** Spaces in variable names (e.g., `daysIn January`) are not allowed in C.
2. **Typos in variable names:** The variable `daysIn CurrentFebrurary` had a spelling mistake (`Februrary` instead of `February`).
3. **Incorrect values for some months:** The number of days for September, October, and November contained invalid expressions such as `3*` and `3+`.
4. **Incorrect sum calculation:** The months were mixed up when computing the number of days in each half of the year.

See Next Page for Corrected Calendar Program Code Implementation

1.2.2 Corrected Calendar Program C Code

```c
#include <stdio.h>

int main() {
    int daysInCurrentFebruary = 29;

    // Define days per month correctly
    int daysInJanuary = 31;
    int daysInFebruary = daysInCurrentFebruary;
    int daysInMarch = 31;
    int daysInApril = 30;
    int daysInMay = 31;
    int daysInJune = 30;
    int daysInJuly = 31;
    int daysInAugust = 31;
    int daysInSeptember = 30;
    int daysInOctober = 31;
    int daysInNovember = 30;
    int daysInDecember = 31;

    // Compute half-year sums
    int daysInFirstHalf = daysInJanuary + daysInFebruary + daysInMarch +
                          daysInApril + daysInMay + daysInJune;
    int daysInSecondHalf = daysInJuly + daysInAugust + daysInSeptember +
                           daysInOctober + daysInNovember + daysInDecember;

    // Print results
    printf("Days in the first half of the current year: %d\n", daysInFirstHalf);
    printf("Days in the second half of the current year: %d\n", daysInSecondHalf);
    printf("Days in the current year: %d\n", daysInFirstHalf + daysInSecondHalf);

    return 0;
}
```

Listing 1.2: Corrected Calendar Program

1.2.3 Quarterly Calendar C Code

```c
#include <stdio.h>

int main() {
    int days[] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    printf("Days in Q1: %d\n", days[0] + days[1] + days[2]);
    printf("Days in Q2: %d\n", days[3] + days[4] + days[5]);
    printf("Days in Q3: %d\n", days[6] + days[7] + days[8]);
    printf("Days in Q4: %d\n", days[9] + days[10] + days[11]);

    return 0;
}
```

Listing 1.3: Quarterly Breakdown Code

- The program above stores all month values in an array, making calculations easier.
- The first six months are summed to get the days in the first half, and the last six months are summed for the second half.
- The total days are correctly computed as 366, accounting for the leap year.
- The quarterly breakdown directly computes and prints the sum of days for each quarter.

1.3 Question 3: Kinetic Energy and Power Calculations

1.3.1 Kinetic Energy Calculation C Code

Formula

The kinetic energy (KE) of the conveyor belt is given by:

$$KE = \frac{1}{2}mv^2 \tag{1.1}$$

where:

- $m = 12.75$ kg (mass of the conveyor belt)
- $v = 3.6$ m/s (velocity of the belt)

C Code for KE formula

```c
#include <stdio.h>

int main() {
    int KE = (int)(0.5 * 12.75 * 3.6 * 3.6);
    printf("Kinetic Energy: %d J\n", KE);

    return 0;
}
```

Listing 1.4: Kinetic Energy Calculation

Explanation

- The program's calculation follows the correct formula $KE = 0.5 \times m \times v^2$.
- The result is 'type-cast' to an integer for microcontroller storage.
- The output is displayed as an integer value in joules (J).

1.3.2 Power and Energy Calculation C Code

Formula

The power (P) and energy (E) are calculated as:

$$P = V \times I \tag{1.2}$$

$$E = P \times t \tag{1.3}$$

where:

- $V = 230V$ (constant voltage)
- $I = 8.5A$ (constant current)
- t (time in hours, provided by the user)

C Code for Power and Energy Calculation

```c
#include <stdio.h>

#define VOLTAGE 230  // Define voltage as a constant

int main() {
    const float current = 8.5;
    unsigned int power = (unsigned int)(VOLTAGE * current);

    float time;
    printf("Enter time in hours: ");
    scanf("%f", &time);

    unsigned int energy = (unsigned int)(power * time);

    printf("Power: %u W\n", power);
    printf("Energy: %u Wh\n", energy);

    return 0;
}
```

Listing 1.5: Power and Energy Calculation

Explanation

- `VOLTAGE` is defined using the preprocessor directive `#define` as `#define VOLTAGE 230`.
- The current value is stored as a constant float (`const float current = 8.5`).
- Power ($P = V \times I$) is computed and stored as an unsigned integer.
- The user inputs the time (`t`), and energy ($E = P \times t$) is calculated.
- The results are printed using the appropriate format specifiers.