Writing Simple Programs

Example 2 (modified)

```
#include<stdio.h>

int main() {
    int height = 8;
    int length = 32;
    int breadth = 16;
    int volume, weight;

    volume = height * length * breadth;
    weight = (volume+165)/166;  // 331/166 = 1 and 332/166 = 2

    printf("weight of the box = %d kg\n", weight);
}
```

Writing Simple Programs

Input

- For input `scanf` is used.
- The value is stored into location (**address**) of the variable.
- For extracting address unary operator `&` is prefixed to variable's identifier.
- Eg. `&height` gives address of variable `height`.

Writing Simple Programs

Example 3

```
#include<stdio.h>
#define FZ_POINT 32.0
#define SCALE_FACTOR  (5.0/9.0)

int main() {
    float farenheit, celsius;

    printf("Enter Farenheit temperature: ");
    scanf("%f", &farenheit);

    celsius = (farenheit - FZ_POINT)*SCALE_FACTOR;

    printf("Celsius equivalent is %1.f\n", celsius);
}
```

Writing Simple Programs

Example 4

```
#include<stdio.h>

int main() {
    int r;
    float pi = 3.1416;
    float volume;

    printf("Enter radius of the sphere; ");
    scanf("%d", &r);

    volume = (4.0/3.0) * pi * r * r * r;

    printf("Volume of the sphere = %.3f\n", volume);

}
```

Writing Simple Programs

How scanf Works

- Like `printf`, `scanf` controlled by format string.
- For each conversion specification, locate item of appropriate type by skipping blanks.
- Read item stopping at character that does not belong to format.
- If read is successful, repeat processing rest of the format.
- It ignores: spaces, form-feed, new-line, tabs.

Writing Simple Programs

How scanf Works (Example 5)

Eg., consider the following `scanf` statement:

```
scanf("%d%d%f%f", &i, &j, &x, &y);

  1
    -20    .3
        -4.0e3
```

- `scanf` will see one continous stream of characters:
  $\sqcup\sqcup 1\backslash n - 20\sqcup\sqcup\sqcup .3\backslash n\sqcup\sqcup\sqcup -4.0\text{e}3\backslash n$
- It will read and skip the characters in sequence: ssrsrrrsssrrsssrrrrrr

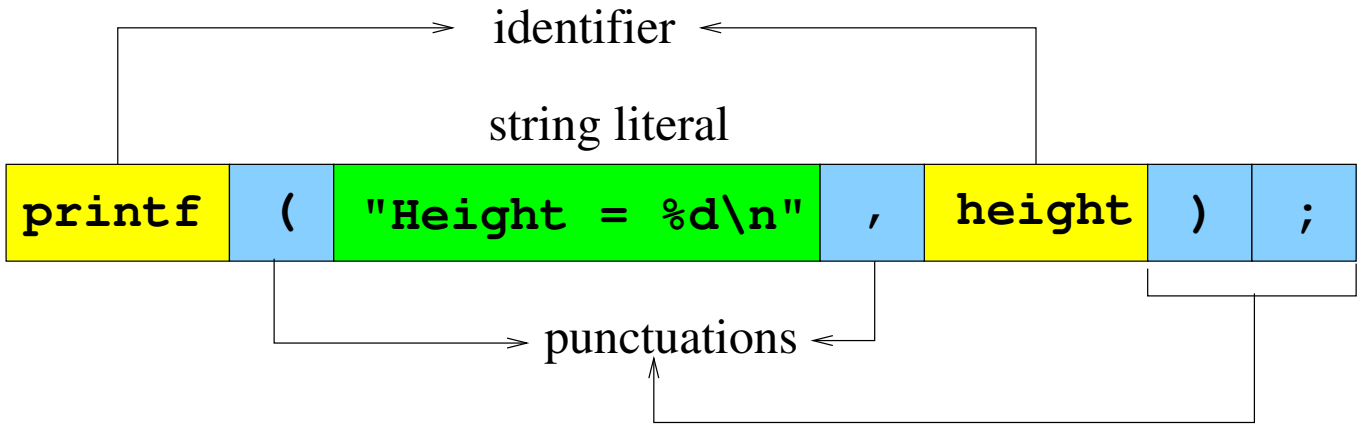
Writing Simple Programs

Format Specifiers

%c	Character
%d	Integer
%i	Integer - same as %d
%f	Float
%e %E	Exponential notation (lowercase/upercase)
%g %G	Uses %f or %E whichever results in shorter
%u	unsigned integer
%o	unsigned octal
%x %X	unsigned hexadecimal (lowercase/upercase)
%p	displays pointer values
%s	Strings

C Tokens

Consists of series of **tokens** separated by space(s).



C Tokens

- Compiler will try to assemble characters into longest possible token.
- So, x–y will be taken as x – y which is illegal.
- So, it is recommended to put white spaces as token separator.
- 15 simple operators: `! , % ^ , & * - + = ~ | . < > / ?`
- 11 compound assignment operators: `( += -= *= /= %= <<= >>= &= ^= != == )`,
- 10 other compound operators: `( -> ++ -- << >> <= >= != && || )`.
- 9 separators: `( ( ) [ ] { } , ; : )`.

Program Readability

- But **indentation** is needed for readability.
- Use of extra space enhances readability.
- But extra space could cause single statement to span multiple lines.
- It is preferable to separate program units by blank lines.
- Choose descriptive names for variable and function identifiers. No maximum limit on length.
- Debugging becomes easy.

Expressions

Arithmetic expressions

- Five operators: `+, -, *, /, %`.
- No exponentiation, but a library function exists: `pow(x,y)`.
- Operands can be integers, floating point numbers or characters.
- Modulus operator need both operands to be integers and second operand to be nonzero.
- Division of integers results in integral quotient.
- `-9 % 7` gives -2, the value `-9/7` is truncated down towards 0.
- Carrying out division with one or both floating point operands then quotient is floating point.

Expressions

Precedence and Associativity

- Relative precedence is:
  - Highest: unary `+`, `-`
  - Higher: binary `*`, `/`, `%`,
  - Lowest: binary `+`, `-`
- Binary operators with equal precedence are left associative.
- Unary operators with equal precedence are right associative.

Expressions

Example 6

Universal Product Code or bar code: defined by 4 group of digits.

- First digit: (0 & 7 for most, 2 for items to be weighed, 3 for medicine/health, 5 for coupons)
- Second group of 5 digits: identifies manufacturers
- Third group of 5 digits: identifies product
- Last digit: check digit.

Expressions

Computing Check Digit

- Add the odd digits: (1-11) call it S_1
- Add the even digits: (2-10) call it S_2
- Find $3 \times S_1 + S_2 - 1$ call it T
- Compute $r = 9 - (T\%10)$
- If $r =$ last digit then the code is read correctly.

Expressions

Check Digit Computation (Example 6)

```
#include <stdio.h>
int main() {
      int first, sum1, sum2, total, result;
      int i1, i2, i3, i4, i5, j1, j2, j3, j4, j5;

      printf("Enter first digit: ");
      scanf("%1d", &first);
      printf("Enter first group: ");
      scanf("%1d%1d%1d%1d%1d", &i1, &12, &i3, &i4, &i5);
      printf("Enter second group: ");
      scanf("%1d%1d%1d%1d%1d", &j1, &j2, &j3, &j4, &j5);

      sum1 = first + i2 + i4 + j1 + j3 + j5;
      sum2 = i1 + i3 + i5 + j2 + j4;
      total = 3 * sum1 + sum2;
      result = 9 - ((total - 1) % 10);
      printf("Check digit is %1d\n", result);
}
```