

## Complete C programs (elementary)

- Operators and Assignment
- Sequence, Selection and Repetition
- Functions
- Pointers
- Getting Input
- References:
  - Introduction to C programming (link on Web site)
  - C reference books (K & R, Afzal)

## Operators and Assignment

## Arithmetic operators

| Operator         | Integer Operands | Floating Point Operands        |
|------------------|------------------|--------------------------------|
| + Addition       | $23 + 10 = 33$   | $22.5 + 10.3 = 32.8$           |
| - Subtraction    | $23 - 10 = 13$   | $22.5 - 10.3 = 12.2$           |
| * Multiplication | $23 * 10 = 230$  | $22.5 * 10.3 = 231.75$         |
| / Division       | $23 / 10 = 2$    | $22.5 / 10.3 = 2.18446(\dots)$ |
| % Remainder      | $23 \% 10 = 3$   | N/A                            |

- Classification
  - Unary Operator: one operand, e.g  $+(6)$ ,  $-(10.43)$ ,  $\sim 12$
  - Binary Operator: two operands, e.g  $23+10$ ,  $23\%10$ ,  $17\sim 14$
  - Ternary Operator: can you find one?

## Relational operators

| Operator | Description         | Example    |
|----------|---------------------|------------|
| >        | Greater than        | $x > 100$  |
| >=       | Greater or equal to | $x >= 20$  |
| <        | Less than           | $x < 100$  |
| <=       | Less or equal to    | $x <= 20$  |
| ==       | Equality            | $x == 100$ |
| !=       | Not equality        | $x != 100$ |

## Assignment statements

- `lvalue = rvalue` e.g. `year = 1900 + 99;`
  - righthand side stored in lefthand side
  - `rvalue` is a variable, constant or expression
  - `lvalue` is a variable
- Assignment is a function returning a value  
`year2 = year = 1968;`
- Minimal type checking! Examples:  
`year = 1968.29; /* year becomes 1968 */`  
`length = 2 / 3; /* length becomes 0.0 */`  
`length = 2./ 3; /* length becomes 0.66.. */`
- `=` is different to the equality operator `==`  
`if (year2 == 1968) ... /* possibly false */`  
`if (year2 = 2001) ... /* always true */`  
`if (year2 = 0) ... /* always false */`

5

COMP2300, 2006

## Precedence, type hierarchy and shorthand notation

- Logical Operators (Afzal 7.8), Bitwise Operators (Afzal Chapter 19)
- Arithmetic Precedence: Table 3.4 in Afzal
- General Operator Precedence: Table 7.7 and D.1 in Afzal
- Data Type Conversion Hierarchy: Table 3.5 Afzal
- Shorthand Operators: Tables 3.7 and 3.8 Afzal
- Increment and Decrement Operators: Tables 3.9-3.14 Afzal

7

COMP2300, 2006

## The Quiz!

- What's the value of the following C expression?  
`7+4*2%6*4`
- Is the following true or false?  
`(16-3<4*3+4) || (7*2==28/2)&&! (16-3<12+6)`
- Are the following all equal?  
`3*4/2`  
`4*3/2`  
`3/2*4`  
`4/2*3`

6

COMP2300, 2006

## String assignments

- String assignment is special:  
`month = "Dec" + "ember"` does not work!
- We have to use the string library `string.h` (see 'man string' on student system)  
`#include <string.h>`  
`...`  
`char month[9]; /* Remember NULL terminator! */`  
`...`  
`strcpy(month, "Dec");`  
`strcat(month, "ember");`

8

COMP2300, 2006

## Sequencing, Selection and Repetition

### Sequencing

- Statements are separated by semicolons ";" and are executed in the order they appear in the function (program code)
- A statement may be a call to a function (usually a different one, but not necessary), and the statements of that function are then executed first

```

...           int doit() {
i = i+1;       ...;
z = doit();   ...;
i = i+2;     return x;
...         }

```

### Selection

- Alternative instruction (sequences) according to some condition(s)

```

if (i == 1) {
    /* Statement sequence 1 */
}
else if (i > 5) {
    /* Statement sequence 2 */
}
else if (i < -42) {
    /* Statement sequence 3 */
}
else {
    /* Statement sequence 4 */
}

```

### Repetition

- Three loop constructs in C:
  - while(condition) { ... }
  - for(start; condition; inc-/decrement) { ... }
  - do { ... } while (condition);
- Use break to exit loops in special cases (Afzal 10.4.1)
- Use continue to jump to bottom of loop (Afzal 10.4.2)
- Never use goto statement!

## while and for loops

```
int main(void) {
    int i;
    ...;
    i = 0;
    while(i<10) {
        printf("%d\n", i);
        i=i+1;
    }
    return 0;
}

int main(void) {
    int i;
    ...;
    for(i=0; i<10; i=i+1) {
        printf("%d\n", i);
    }
    return 0;
}
```

## Functions

## Functions

- Functions must be **declared** before they are used. The **definition** (i.e. **implementation**) of the function can come later in the program (or even in a different module)
  - For example `#include <stdio.h>` for `printf()` definition
- A function declaration consists of:
 

```
return_type function_name(parameter_list);
```

 Examples:
 

```
int add(int x, int y);
int main(void);
void display(float number);
```
- Parameters are **passed-by-value**, so they won't be changed. They are essentially variables locally to the function, which are initialised when it is called

## Example simple\_function.c

```
1: #include <stdio.h>
2: int next_year(int);
3: int main(void) {
4:     int a,b;
5:
6:     a=1981;
7:     b=next_year(a);
8:
9:     if (b==1982) {
10:        printf("1982\n");
11:    }
12:    else {
13:        printf("Help!\n");
14:    }
15:    return 0;
16: }
17:
18: int next_year(int y) {
19:     int new_year;
20:
21:     y=y+1;
22:     /* Change y, but not a! */
23:
24:     new_year=y;
25:
26:     return new_year;
27: }
```

## Pointers! (Afzal Chapter 4)

### Five Attributes of Variables

- Type
- Name
- Contents
- Size
  - The `sizeof()` operator gives the size (in bytes)
    - `sizeof(data type)`
    - `sizeof(variable name)`
- Address (or a pointer in C)

## Pointers

- *A pointer is something which may or may not exist*  
Anonymous
- A pointer is a variable containing a memory address, usually this is the address of another (non-pointer) variable
- We can use the memory address to read or modify the value of the variable the pointer refers to
  - `int *` Type pointer to int
  - `*x` Variable pointed to by x
  - `&y` Address of y
- Why are these not valid?
  - `&100` `&'A'` `&(K+2)`

### Pointer example

```

1: int main(void) {
2:     int a = 2;
3:     int *x; /* x is a pointer to a variable of type int */
4:
5:     x = &a; /* The address of a is assigned to x */
6:           /* x points now to where a is in memory */
7:
8:     *x = *x+1; /* Increment the variable that x points */
9:           /* to, i.e. the variable a */
10:
11:    printf("a=%d\n", a);
12:    printf("*x=%d\n", *x);
13:    printf("x=%p\n", x); /*Note pointer format specification*/
14:    printf("&a=%p\n", &a);
15:
16:    return 0;
17: }
```

## Pointers as parameters

- Passing a pointer as a parameter to a function, allows the function to change the variable the pointer refers to
- This allows us to work around the restriction that a function cannot modify any variables (other than those local to it)
- We don't change a pointer, only the variable it points to
- Remember that all variables are **passed-by-value**, i.e. the parameter values are copied to local variables when a function is called. A memory address is just a value, so a copy of a pointer is just as good as the original pointer itself.

## Getting Input

## Pointer example (2)

```

1: #include <stdio.h>           1: void next_year(int *y) {
2:                               2:
3: void next_year(int *y);      3:     *y = *y + 1 ;
4:                               4:
5: int main(void) {             5: }
6:     int this_year;
7:
8:     this_year = 1981;
9:     next_year(&this_year);
10:
11:    printf("Year: %d\n", this_year);
12:
13:    return 0;
14: }
```

## Getting input – scanf()

- The `stdio` library provides functions to read values from keyboard and other sources (like files)
 

```
int scanf(const char *format, ...);
```
- The `format` string contains conversion specifications indicating how many, and what type of values to read
- The subsequent parameters are pointers to the variables in which `scanf()` should store the values
 

```
Example: scanf("%d", &year);
```
- `scanf()` uses whitespace characters (return, tab and space) to decide how to divide the input into separate fields

## scanf() – Example

```
1: #include <stdio.h>
2:
3: int main(void) {
4:     int a, b;
5:
6:     scanf("%d %d", &a, &b);
7:
8:     printf("sum=%d\n", a+b);
9:
10:    return 0;
11: }
```

## For Next Lecture

```
1: #include <stdio.h>
2: int main(){
3:     char name[100];
4:     printf(" Enter your name\n");
5:     scanf("%s",name);
6:     printf("Your name is %s\n",name);
7:     return 0;
8: }
```

- Why is there no & in this scanf()?
- Run the program and input both your first and lastname separated by a space. What is printed out and why?