Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CS Basics
9) C Prog. Lang.

E. Benoist & C. Grothoff

Fall Term 2018-19

Computer Science Division

The C Programming Language

Introduction

Introduction

- **C was developed by Brian Kernighan and Denis Ritchie**
  - At the Bell Labs in the 1970s
  - They published a definitive description of the language (1978)
  - Version of the language called K&R C
- **ANSI C**
  - In the 1980s: different incompatible versions of C
  - ANSI developped a standardized version of C
  - Most of the compilers adhere to the standard

Structure of a program

- **A C program consists of one or more functions**
  - One function must be called `main`
  - The program allways begins by executing `main`
- **Each function must contain**
  - A function heading, (function name, followed by list of arguments)
  - List of argument declarations
  - A compound statement

Hello World

Hello World Program

- **helloworld.c**

```c
#include <stdio.h>

main(){
  printf("Hello World\n");
}
```


Square

- **helloworld2.c**

```c
#include <stdio.h>

main(){

  printf("Hello, compute the square of a ↘
  →number\n");
  printf("What number?\n");
  int number; // We reserve memory
  scanf("%d", &number); // We set the value ↘
  →inside memory
  printf("The square of %d is %d\n",number,↘
  →number*number);

}
```

C Fundamentals

Data Types

Data Types

Typical memory requirements:

- `int` Integer quantity
  2 bytes or 4 bytes (depending on the platform)
- `char` single character
  1 byte (8 bits)
- `float` floating-point number
  4 bytes (32 bits)
- `double` double precision floating-point number
  8 bytes (64 bits)

For many C types, details depend on the target platform!

Constants

Integer constants

- **Decimal**
  - 0
  - 1
  - -743
  - 5280
- **Ocatal**
  - 01
  - 0654
  - 07777
- **Hexadecimal**
  - 0xC
  - 0x12AC
  - 0xFFFF
  - 0x10000

Example

- **constants.c**

```
int i1 = 12345;
int i2 = 0;
int i3 = -145;
int i4 = 234;
puts("Integer values");
printf("i1 = %d, i2=%d, i3=%d, i4=%d\n",i1,i2,i3,i4)↘
→;
int j1 = 0;
int j2 = 017;
int j3 = 07777;
printf("j1 = %d, j2=%d, j3=%d\n",j1,j2,j3);
/* Output: j1 = 0, j2=15, j3=4095  */
int k1 = 0xA;
int k2 = 0x10;
int k3 = -0xFFFF;
int k4 = 0x10000;
printf("k1=%d, k2=%d, k3=%d, k4=%d\n",k1,k2,k3,k4);
/* Output: k1=10, k2=16, k3=-65535, k4=65536 */
puts("(or written in hexadecimal):");
printf("k1=%x, k2=%x, k3=%x, k4=%x\n",k1,k2,k3,k4);
/* Output: k1=a, k2=10, k3=ffff0001, k4=10000 */
```


Unsigned and Long Integers

- **Normal `int`s are noted in 2's complement**
  - Typically use 4 bytes (i.e. 32 bits)
  - Maximal number = 0x7FFFFFFF (i.e. 2'147'483'647), use `INT_MAX`
  - Minimal number =-0x80000000 (i.e. -2'147'483'647), use `INT_MIN`
- **Unsigned integers can only be positive**
  - Minimal unsigned int = 0
  - Maximal unsigned int = 0xFFFFFFFF, use `UINT_MAX`
- **Long integers are typically 8 bytes**
  - `long long` go until 0x7FFFFFFFFFFFFFFF (and also negative), use `LLONG_MAX`
  - And `unsigned long long` go until 0xFFFFFFFFFFFFFFFF, use `ULLONG_MAX`

Example

- **constants.c (cont.)**

```
 /* Largest integer */
int k5 = 0x7FFFFFFF;
/* Smallest integer */
int k6 =-0x80000000;
printf("MAXINT= %x, (%d), MININT=%x,(%d) \n↘
→",k5,k5,k6,k6);

long l1 = 10;
long l2 = -20;
long l3 = 0xFFFFFFFFFFF;
printf("l1 = %ld, l2=%ld, l3=%ld\n",l1,l2,↘
→l3);

unsigned long ul1 = 0xFFFFFFFFFFFFFFFF;
long l4=-1;
printf("ul1 = %ld, l4=%ld\n",ul1,l4);
/* Output : ul1 = -1, l4=-1  */
```

Defined bit width via `stdint.h`

- `int8_t`
- `int16_t`
- `int32_t`
- `int64_t`
- `uint8_t`
- `uint16_t`
- `uint32_t`
- `uint64_t`

Use these if you need a particular range!

Floating points

- **Two types**
  - `float`: floating point numbers (4 bytes, mantiassa 23 bits)
  - `double`: double precision floating point numbers (8 bytes, mantissa 52 bits)
- **Using . notation**
  - float f1 = 0.899;
  - -9.788
  - 0.00001
- **Using scientific notation**
  - 0.01E3
  - 1.999E-5
  - -10.999E3

Characters

- **Single characters enclosed in apostrophes**
  - char x = 'A';
  - 'x'
  - '3'
  - '?'
- **Character can be seen as numbers (ASCII code)**
  - 'A' = 65 = 0x41
  - '0' = = 0x30
- **Spectial non viewable characters are represented by escape sequence**
  - `'\n'` = 10 = 0xA (newline)
  - `'\t'` (tabulation)
  - `'\0'` (end of string in C)

Strings

- **Constant strings are written with (double) quotation marks**
  - `"green"`
  - `"Hello World"`
  - `"Type 1 to continue \n"`
- **In C, strings are '\0'-terminated arrays.**

Variables

Variables

- **Must be declared**
  - Each variable belongs to one type
  - `int a, b, c;`
  - `double d1, d2, d3;`
- **The value of the variable must be initialized**
  - `a=100;`
  - `b=0;`
  - `c=0x100;`
  - `d1=5.0;`
  - `d2=4E3;`
  - `d3=d1+d2*c;`
- **Values can be modified**
  - `a=200;`
  - `a=2*a+b*c;`

Char und Strings declaration

- **Characters**

```
char c1, c2;
c1 = '4';
c2 = 'A';
char c3 = '\n';
```

- **String are arrays of characters**

```
char text[80]; // a string of 80 characters
char text[] = "Biel/Bienne";
```

- **Be careful to reserve enough space**

```
// will lose half of the name
char city[10]="Ville de Bienne";
// The \0 is lost
char city2[15]="Ville de Bienne";
```


Symbolic Constants

Symbolic Constants

- **A name that substitutes a sequence of characters**
  - The characters may represent a numeric constant, a character constant or a string constant
  - When the program is compiled, each occurrence of the name is replaced by the constant
  - Syntax

```
#define name text
```

- **Example**

```
#define PI 3.141593
#define TRUE 1
....
area = PI * radius * radius
```


Operators and Expressions

Arithmetic Operators

Arithmetic Operators

- **Five arithmetic binary operators**
  - + addition
  - - substraction
  - * multiplication
  - / division
  - **%** modulo, the remainder after integer division
- **Unary operators**
  - - (unary minus operation) For generating negative numbers

```
-756      -9      -0x99A
```

  - **increment operator ++**
    Can be placed before or after the variable, variable is incremented before or after it is utilized

```
int i1 = 35;
printf("Value of i1=%d", i1);
printf("Value of ++i1=%d", ++i1); // 36
printf("Value of i1++=%d", i1++); // 36
printf("Value of i1=%d",i1); // 37
```

Operator sizeof

- **Returns the number of bytes used to store a variable**

```
// Operator sizeof
int j1;
long l1;
float f1;
double d1;
char str1[10]="Biel";
printf("size of j1=%ld\n",sizeof j1);
printf("size of l1=%ld\n",sizeof l1);
printf("size of f1=%ld\n",sizeof f1);
printf("size of d1=%ld\n",sizeof d1);
printf("size of str1=%ld\n",sizeof str1);
```

- **More useful: Length of a string**

```
char text[]="Canton of Bern";
printf("size of text=%ld\n",sizeof text);
```


Relational and logical operators

- **Relational operators**
  - <, <=, >,>=
  - == equal to
  - != not equal to
  - Are used to form logical expressions: true (i.e. not 0) or false (i.e. 0)
- **Logical Operators**
  - || or
  - && and

```
i1=20;
i2=22;
if(i1<30 || i1>i2){
  puts("condition is true");
}
else{ // should not arrive, since i1<30
  puts("condition is false");
}
```


Assignment Operators

- **Used to assign the value of an expression to an identifyer**
  - `identifier = expression`

```
int i1;
int i2 = 4;
i1 = 3;
i1 = i2 * 4;
i1 = i2 + i1 + 5;
```

- **Multiple assignement is legal in C**
  - `identifier1 = identifier2 = identifier3`
  - is equivalent to
  - `identifier1 = (identifier2 = identifier3)`

```
int i, j;
i = j = 5;
printf("i=%d, j=%d\n",i,j);
// Output: i=5, j=5
```


More Assignment Operators

- **Five additional assignment operators**
  - +=, -=,*=,/= and %=
- **Syntax**
  - if i3 is an integer variable
  - `i3 += 5;`
  - is equivalent to
  - `i3 = i3 + 5;`

```
int i=0;
i += 5; // i is 5
i *= 2; // i is 10
printf("i=%d",i); // i=10
```

The conditional operator

- **expression 1 ? expression 2 : expression 3**
  - if expression 1 is evaluated
  - if expression 1 is true, expression 2 is evaluated and returned
  - else expression 3 is evaluated and returned
- **Example**

```
int i4 = (i3<0) ? 0 : 100;
 printf("i4=%d\n",i4);
 float f1 = 1.5;
 float f2 = 2.3;
 float fmin = (f1<f2) ? f1 : f2;
```


Library functions

Library functions

- **Libraries contain useful functions**
  - Standard libraries
  - Self made libraries
- **For dealing with Input / Output** `stdio.h`
  - Scan input from stdin
  - Write output to stdout
  - Open and close files
  - Write output to file
  - Read files
  - Format strings
- **For Mathematics** `math.h`
  - Compute mathematical functions
  - Knows the mathematical constants

Example with stdio.h

- **library.c**

```
#include <stdio.h>
main(){
  char name [26];
  puts("What␣is␣your␣name?");
  scanf("%s",name);
  printf("Hello␣%s␣\n",name);
}
```

Example with math.h and stdio.h

- **libary2.c** needs to be linked using **-lm**:

```
gcc library2.c -lm -o library2
```

```
#include <stdio.h>
#include <math.h>
main(){
  float x, y;
  double logX, xPowerY;
  puts("Type␣an␣integer␣x?");
  scanf("%f",&x); // argument is address of ↘
  →x
  puts("Type␣an␣integer␣y?");
  scanf("%f",&y); // address of y
  logX = log(x)/log(2); // since log(x) is ↘
  →the natural logarithm
  printf("Log␣of␣%f␣=␣%f␣\n",x,logX);
  xPowerY = pow(x,y);
  printf("%f␣power␣%f␣=␣%f␣\n",x,y,xPowerY);
}
```


Conclusion

Conclusion

- **Data Types**
  - Each variable has one type
  - The value in memory is evaluated using this type
  - `int`: signed integers on 4 bytes (can also be `unsigned`)
  - `long long`: signed integers on 8 bytes (can also be `unsigned`)
  - `float`, `double`
  - `char` and `char[]` (i.e. strings)
- **Symbolic constants**
  - Very efficient: replaced at pre-compilation step
  - Do never write a constant inside your code (string / number)
- **Libraries**
  - `stdio.h` : for input / output
  - `math.h` : for mathematic functions
  - `stdlib.h` : for standard functions (manipulation of strings for instance)

Bibliography

- This lecture corresponds to chapters *1*, *2* and *3* of the course book:
  Schaum's OuTlines, Programming with C (second edition), *Byron Gottfried*, Mc Graw-Hill, 1996