

Introduction to Computer and Programming

Chapter 6: Basic C

Manuel

Fall 2018

Outline

- 1 Data types
- 2 Structures
- 3 Control statements

Types of variables

Three main categories of variables:

- Constant variables: `#define PI 3.14159`
- Global variables: defined for all functions
- Local variables: defined only in the function

Types of variables

Three main categories of variables:

- Constant variables: `#define PI 3.14159`
- Global variables: defined for all functions
- Local variables: defined only in the function

Never ever use global variables in VG101

Initialising variables

Common use:

- Variables for `#define` are UPPERCASE
- Other variables are lowercase, or capitalised
- Variable names cannot exceed 31 characters
- Variable names can start with `_` or a character
- Variables starting with `_` are “hidden”

Basic data types

Data types in C:

- Integer: `int`
- Character: `char`
- Valueless type: `void`
- Fractional numbers:
 - Single precision: `float`
 - Double precision: `double`

The C standard only fixes the size of `char` (1 byte)

Basic data types

Data types in C:

- Integer: `int`
- Character: `char`
- Valueless type: `void`
- Fractional numbers:
 - Single precision: `float`
 - Double precision: `double`

The C standard only fixes the size of `char` (1 byte)

Different variations available:

- `char`: signed `char`, unsigned `char`
- `int`: short `int`, signed short `int`, unsigned short `int`, signed `int`, unsigned `int`, long `int`, signed long `int`, unsigned long `int`, long long `int`, signed long long `int`, unsigned long long `int`
- `double`: long `double`

Extra variations: `static`, `register`, `extern`, `volatile`

Data types

Basic number types:

- int: size limitation, from 0 to $2^{32} - 1$
- float: 7 digits of precision, from $1 \cdot 10^{-38}$ to $3 \cdot 10^{38}$
- double: 13 digits of precision, from $2 \cdot 10^{-308}$ to $1 \cdot 10^{308}$

Example.

```
1 float a=1.0; int b=3; double c;
```


Data types

Basic number types:

- `int`: size limitation, from 0 to $2^{32} - 1$
- `float`: 7 digits of precision, from $1 \cdot 10^{-38}$ to $3 \cdot 10^{38}$
- `double`: 13 digits of precision, from $2 \cdot 10^{-308}$ to $1 \cdot 10^{308}$

Example.

```
1 float a=1.0; int b=3; double c;
```

Characters:

- No type for strings, only for single characters
- Strings are viewed as arrays of characters
- Characters are enclosed in single quotes, e.g. `char a='a';`
- Strings are enclosed in double quotes
- Character are encoded using the American Standard Codes for Information Interchange (ASCII)

Wrong data type

What output to expect?

types1.c

```
1  #include <stdio.h>
2  int main() {
3      printf("%d %f\n", 7/3, 7/3);
4  }
```

Wrong data type

What output to expect?

types1.c

```
1  #include <stdio.h>
2  int main() {
3      printf("%d %f\n", 7/3, 7/3);
4  }
```

Fixed version:

types2.c

```
1  #include <stdio.h>
2  int main() {
3      printf("%d %f\n", 7/3, 7.0/3);
4      int a=42; char b=(char) a;
5      printf("%c\n", b);
6  }
```

Wrong data types

Understanding the code:

- What do %f, %d and %c mean?
- What is the type of 7/3 for the compiler?
- What is displayed for b?
- What is this character corresponding to?
- Why is this character displayed?

Type casting

Changing data type:

- Float to int: `float a=4.8; int b= (int) a;`
- Int to char: `int a=42; char b=(char) a;`
- Try double to char, int to float

Type casting

Changing data type:

- Float to int: `float a=4.8; int b= (int) a;`
- Int to char: `int a=42; char b=(char) a;`
- Try double to char, int to float

Always think of the size...

Example

types3.c

```
1  #include <stdio.h>
2  int main() {
3      float c=4.8; printf("%d\n", (int)c);
4      int f=42; printf("%c\n", (char)f);
5      double a=487511234.7103;
6      char b=(char) a;
7      printf("%c, %c\n",b,a);
8      int d=311;
9      float e=(float) d;
10     printf("%d %f\n",d,e);
11     printf("%c\n",d);
12 }
```

Example

Understanding the code:

- Which type casting work well?
- What is the length of a `char`?
- What is the length of an `int`?
- What is printed for `d`?
- What is the issue when displaying `d` as a `char`?

Example

Write C program featuring a function `apbp1(float a, float b)` which returns the nearest integer to $a+b+1$.

Example

Write C program featuring a function `apbp1(float a, float b)` which returns the nearest integer to `a+b+1`.

apbp1.c

```
1  #include <stdio.h>
2  int apbp1 (float a, float b);
3  int main () {
4      float a, b;
5      scanf("%f %f", &a,&b);
6      printf("%d\n", apbp1(a,b));
7  }
8  int apbp1 (float a, float b) {
9      a++; a+=b;
10     return((int) (a+0.5));
11 }
```

Question. How are shorthand operators and type casting used?

Outline

① Data types

② Structures

③ Control statements

What, why data types?

More data types in C:

- Reminder: a bit belongs to $\{0, 1\}$ and a byte is 8 bits
- Operating data at low level, e.g. shift `<<`, `>>`
- A char does not necessarily contains a character
- Logical operations are of a major importance
- Understanding data representation is important to be efficient
- Structures, enumerate, union

Structures

st.c

```
1  #include <stdio.h>
2  typedef struct _person {
3      char* name;
4      int age;
5  } person;
6  int main () {
7      person al={"albert",32};
8      person gil;
9      gil.name="gilbert";
10     gil.age=23;
11     struct _person so={"sophie",56};
12     printf("%s %d\n",al.name, al.age);
13     printf("%s %d\n",gil.name, gil.age);
14     printf("%s %d\n",so.name, so.age);
15 }
```

Structures

Understanding the code:

- How is a structure defined?
- How to define a new type?
- What are two ways to set the value of a field in a structure?
- How to access the values of the different fields in a structure?

Functions and structures

st-fct.c

```
1  #include <stdio.h>
2  typedef struct person {
3      char* name; int age;
4  } person_t;
5  person_t older(person_t p, int a);
6  int main () {
7      person_t al={"albert",32};
8      al=older(al,10);
9      printf("%s %d\n",al.name,al.age);
10 }
11 person_t older(person_t p, int a) {
12     printf("%s %d\n",p.name, p.age);
13     p.age=p.age+a;
14     return p;
15 }
```

Functions and structures

Understanding the code:

- How is the age increased?
- How are the person's information sent to a function?
- How to return the person's information after the function?
- How many output can a C function have?

Outline

- ① Data types
- ② Structures
- ③ Control statements

Jumping!

jump.c

```
1  #include <stdio.h>
2  int main() {
3      int i=0;
4      printf("I am at position %d\n",i);
5      i++;
6      goto end;
7      printf("I am at position %d\n",i);
8      end:
9          i++;
10         printf("It all ends here, at position %d\n",i);
11         return 0;
12         i++;
13         printf("Unless it's here at position %d\n",i);
14     }
```

Jumping!

Understanding the code:

- What positions are displayed?
- Why are some positions skipped?
- How to use the goto statement?
- Why should the goto statement (almost) never be used?

Important operators

Basics on conditional statements:

- No boolean type, 0 means False, anything else True
- Boolean evaluation: <, <=, >, >=, ==, !=
- Not: !, short-circuit operators: &&, or: ||
- Bit operations: &, |, ^

Important operators

Basics on conditional statements:

- No boolean type, 0 means False, anything else True
- Boolean evaluation: <, <=, >, >= , ==, !=
- Not: !, short-circuit operators: &&, or: ||
- Bit operations: &, |, ^

Conditional ternary operator: ?:

```
1 condition ? expression1 : expression2
```

Important operators

Basics on conditional statements:

- No boolean type, 0 means False, anything else True
- Boolean evaluation: <, <=, >, >=, ==, !=
- Not: !, short-circuit operators: &&, or: ||
- Bit operations: &, |, ^

Conditional ternary operator: ?:

```
1 condition ? expression1 : expression2
```

Example.

A macro returning the max of two numbers:

```
1 #define MAX(a,b) a>=b ? a : b
```

The if and switch statements

```
1  if (condition) {  
2      statements;  
3  }  
4  else {  
5      statements;  
6  }
```

```
1  switch(variable) {  
2      case value1:  
3          statements;  
4          break;  
5      case value2:  
6          statements;  
7          break;  
8      default:  
9          statements;  
10         break;  
11 }
```

Example

cards.c

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<time.h>
4  #define ACE 14
5  #define KING 13
6  #define QUEEN 12
7  #define JACK 11
8  int main () {
9      int c;
10     srand(time(NULL)); c=rand()%13+2;
11     switch (c) {
12         case ACE: printf("Ace\n"); break;
13         case KING: printf("King\n"); break;
14         case QUEEN: printf("Queen\n"); break;
15         case JACK: printf("Jack\n"); break;
16         default: printf("%d\n",c); break;
17     }
18 }
```


Example

Understanding the code:

- Write this code using the `if` statement
- Adapt the code such as to display the complete card name (e.g. “Ace of spades”)
- What happens if a `break` is removed?
- Explain why and compare to the behavior in MATLAB

The while and do... while statements

Structure of a while loop:

```
1 while (conditions) {  
2     statements;  
3 }
```

Structure of a do... while loop:

```
1 do {  
2     statements;  
3 } while (conditions);
```

The while and do... while statements

Structure of a while loop:

```
1 while (conditions) {  
2     statements;  
3 }
```

Structure of a do... while loop:

```
1 do {  
2     statements;  
3 } while (conditions);
```

Example.

```
1 int i=0;  
2 while(i++<3) {  
3     printf("%d",i);  
4 }
```

```
1 int i=0;  
2 do {  
3     printf("%d",i);  
4 } while(i++<3);
```

Questions.

- What is the difference between the two outputs?
- What happens if ++i and i++ are changed?

The for statement

Structure of a for loop:

```
1 for(init;test;step) { statements; }
```

- init: executed at the beginning of the loop
- test: tested at the beginning of each iteration
- step: executed at the end of each iteration

The for statement

Structure of a for loop:

```
1  for(init;test;step) { statements; }
```

- init: executed at the beginning of the loop
- test: tested at the beginning of each iteration
- step: executed at the end of each iteration

Example.

```
1  for(i=0; i<n; i++)  
2      printf("%d ", i);  
3  i=0; for(;i<n;i++)  
4      printf("%d ", i);  
5  for(i=0; i<n;)  
6      {printf("%d\n",i); i++;}  
7  for(i=0;i<n;)  
8      printf("%d ",i++);
```

```
1  fct=1;  
2  for(i=1;i<=n;i++) fct*=i;  
3  printf("%d ", fct);  
4  for(i=1,fct=1;i<=n;fct*=i,i++);  
5  printf("%d ", fct);  
6  for(i=1,fct=1;i<=n;fct*=i++);  
7  printf("%d\n", fct);
```

The break and continue statements

Questions.

- What are the loops on the right doing?
- How is the code indented
- Which for loop is the clearest and best used?

The break and continue statements

Questions.

- What are the loops on the right doing?
- How is the code indented
- Which for loop is the clearest and best used?

Acting from within a loop:

- Early exit of a loop: break
- Skip to the next loop iteration: continue

Example.

```
1  for(i=0;i<10;i++) {  
2      scanf("%d",&n);  
3      if(n==0) break;  
4      else if(n>=10) continue;  
5      printf("%d\n", n);  
6  }
```

Key points

- What are the main data types in C?
- How to perform type casting?
- How to define and use structures on C?
- How to perform conditional statements in C?
- How to write loops in C?

Thank you!