Arithmetic types of C language

Basics of Programming 1

DEPARTMENT OF
NETWORKED SYSTEMS
AND SERVICES

G. Horváth, A.B. Nagy, Z. Zsóka, P. Fiala, A. Vitéz

7 October, 2020

Content

Chapter 1

Functions

Segmentation – motivation

Let's create a program, that prints out the sum of the squares of all positive numbers, that are smaller than 12! $(1^2 + 2^2 + \cdots + 11^2)$

Segmentation – motivation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's create a program, that prints out the sum of the squares of all positive numbers, that are smaller than 12! $(1^2 + 2^2 + \cdots + 11^2)$

```c
#include <stdio.h> /* for printf */

int main(void)
{
  int i, sum; /* aux. variable and sum of squares*/

  sum = 0;                           /* initialization */
  for (i = 1; i < 12; i = i+1) /* i = 1,2,...,11 */
    sum = sum + i*i;                 /* summing */

  printf("The square sum: %d\n", sum);
  return 0;
}
```

link

Segmentation – motivation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;            /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;            /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;            /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!

Segmentation – motivation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;            /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;            /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;            /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!

Our solution

- was made by Copy+Paste+correct

Segmentation – motivation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;            /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;            /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;            /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!

Our solution

- was made by Copy+Paste+correct
- many possibilities for mistakes, errors

Segmentation – motivation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;            /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;            /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;            /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!

Our solution

- was made by Copy+Paste+correct
- many possibilities for mistakes, errors
- long program

Segmentation – motivation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;            /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;            /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;            /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!

Our solution

- was made by Copy+Paste+correct
- many possibilities for mistakes, errors
- long program
- it is hard to manage

Segmentation – motivation


```c
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;            /* for 12 */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;            /* for 24 */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;            /* for 30 */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

link

Let's create a program, that will perform the previous tasks with numbers 12, 24 and 30!

Our solution

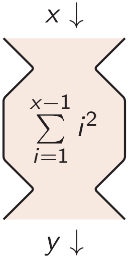
- was made by Copy+Paste+correct
- many possibilities for mistakes, errors
- long program
- it is hard to manage

Is it possible in a more smarter way?

Functions

The function

- Standalone program segment
- For operations that occur frequently
- We can run it (call it) with different arguments
- Calculates something, and gives back the result for the program that called it



Functions – solution

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
int squaresum(int n) /* function definition */
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}

int main(void) /* main program */
{
  int sum1, sum2, sum3;

  sum1 = squaresum(12); /* function call */
  sum2 = squaresum(24);
  sum3 = squaresum(30);

  printf("%d, %d, %d\n", sum1, sum2, sum3);
  return 0;
}
```

link

Function definition

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of a function definition

<type of return value>
<function identifier> (<list of formal parameters>)
<block>

```
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of a function definition

<type of return value>
<function identifier> (<list of formal parameters>)
<block>

```
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of a function definition

<type of return value>
<function identifier> (<list of formal parameters>)
<block>

```
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

Syntax of a function definition

<type of return value>
<function identifier> (<list of formal parameters>)
<block>

```
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of a function definition

```
<type of return value>
<function identifier> (<list of formal parameters>)
<block>
```

```c
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of a function definition

<type of return value>
<function identifier> (<list of formal parameters>)
<block>

```
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Function definition

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Type of the return value:

- The type of the calculated value

```
double average(int a, int b)
{
  return 0.5 * (a + b);
}
```

Function definition

Type of the return value:

- The type of the calculated value

```
double average(int a, int b)
{
  return 0.5 * (a + b);
}
```

- or void (empty), if the function does not calculate anything

```
void print_point(double x, double y)
{
  printf("(%.3f, %.3f)", x, y); /* (2.000, 4.123) */
}
```

Function definition

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Type of the return value:

- The type of the calculated value

```
double average(int a, int b)
{
  return 0.5 * (a + b);
}
```

- or `void` (empty), if the function does not calculate anything

```
void print_point(double x, double y)
{
  printf("(%.3f, %.3f)", x, y); /* (2.000, 4.123) */
}
```

- because sometimes we don't care about the calculated value, only about the "side effect" (secondary effect).

A remark: Primary and secondary effects

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Primary the function calculates and gives back the return value

A remark: Primary and secondary effects

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Primary the function calculates and gives back the return value

Secondary the function "performs some more things" (prints on screen, writes to file, plays an MP3, launches a missile. . .)

A remark: Primary and secondary effects

Primary the function calculates and gives back the return value

Secondary the function "performs some more things" (prints on screen, writes to file, plays an MP3, launches a missile. . .)

- Some programming languages make a clear distinction between different program segments:

A remark: Primary and secondary effects

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Primary the function calculates and gives back the return value

Secondary the function "performs some more things" (prints on screen, writes to file, plays an MP3, launches a missile. . .)

- Some programming languages make a clear distinction between different program segments:
  - **function** where the primary effect is the important

A remark: Primary and secondary effects

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Primary the function calculates and gives back the return value

Secondary the function "performs some more things" (prints on screen, writes to file, plays an MP3, launches a missile...)

- Some programming languages make a clear distinction between different program segments:
  - **function** where the primary effect is the important
  - **procedure** no primary effect, but the secondary effect is important

A remark: Primary and secondary effects

Primary the function calculates and gives back the return value

Secondary the function "performs some more things" (prints on screen, writes to file, plays an MP3, launches a missile. . .)

- Some programming languages make a clear distinction between different program segments:
  - **function** where the primary effect is the important
  - **procedure** no primary effect, but the secondary effect is important
- In C language there is only function. Procedures are represented by functions with empty (`void`) return value.

A remark: Primary and secondary effects

Primary the function calculates and gives back the return value

Secondary the function "performs some more things" (prints on screen, writes to file, plays an MP3, launches a missile. . .)

- Some programming languages make a clear distinction between different program segments:
  - **function** where the primary effect is the important
  - **procedure** no primary effect, but the secondary effect is important
- In C language there is only function. Procedures are represented by functions with empty (`void`) return value.
- Generally, we should try to separate the primary and secondary effects!

Function definition

Formal list of parameters

- Comma-separted list of declaration of parameters one-by-one, so we can reference them inside the function

```
double volume(double x, double y, double z)
{
  return x*y*z;
}
```

Function definition

Formal list of parameters

- Comma-separted list of declaration of parameters one-by-one, so we can reference them inside the function

```
double volume(double x, double y, double z)
{
  return x*y*z;
}
```

- The number of parameters can be 0, 1, 2, . . . as much as you want (127 )

Function definition

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Formal list of parameters

- Comma-separted list of declaration of parameters one-by-one, so we can reference them inside the function

```
double volume(double x, double y, double z)
{
  return x*y*z;
}
```

- The number of parameters can be 0, 1, 2, . . . as much as you want (127 ☺)
- If there are 0 parameters, we denote it with `void`

```
double read_next_positive(void)
{
  double input;
  do scanf("%lf", &input) while (input <= 0);
  return input;
}
```

Function definition

The `return` statement

- it gives a return value, it terminates the execution of the function's block, and returns to the point of calling

Function definition

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

The return statement

- it gives a return value, it terminates the execution of the function's block, and returns to the point of calling
- there can be more of it, but it will cause to (terminate and) return to the point of calling at the first execution.

```
double distance(double a, double b)
{
  double dist = b - a;
  if (dist < 0)
    return -dist;
  return dist;
}
```

Function definition

The `return` statement

- it gives a return value, it terminates the execution of the function's block, and returns to the point of calling
- there can be more of it, but it will cause to (terminate and) return to the point of calling at the first execution.

```
double distance(double a, double b)
{
  double dist = b - a;
  if (dist < 0)
    return -dist;
  return dist;
}
```

- it can also occur in a `void`-type function `return;`

Function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
double distance(double a, double b)
{
   ...
}
```

Syntax of a function call

<function identifier> (<actual argument expr.>)

```c
double x = distance(2.0, 3.0); /* x will be 1.0 */
```

Function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
double distance(double a, double b)
{
   ...
}
```

Syntax of a function call

<function identifier> (<actual argument expr.>)

```c
double x = distance(2.0, 3.0); /* x will be 1.0 */
```

Function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
double distance(double a, double b)
{
   ...
}
```

Syntax of a function call

<function identifier> (<actual argument expr.>)

```
double x = distance(2.0, 3.0); /* x will be 1.0 */
```

Function call

```c
double distance(double a, double b)
{
   ...
}
```

Syntax of a function call

<function identifier> (<actual argument expr.>)

```c
double x = distance(2.0, 3.0); /* x will be 1.0 */
```

```c
double a = 1.0;
double x = distance(2.5-1.0, a); /* x will be 0.5 */
```

Function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
double distance(double a, double b)
{
    ...
}
```

Syntax of a function call

<function identifier> (<actual argument expr.>)

```c
double x = distance(2.0, 3.0); /* x will be 1.0 */
```

```c
double a = 1.0;
double x = distance(2.5-1.0, a); /* x will be 0.5 */
```

```c
double pos = read_next_positive(); /* empty () */
```

The main program as a function

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
int main(void) /* now we understand, what this is */
{
  ...
  return 0;
}
```

The main program as a function

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
int main(void) /* now we understand, what this is */
{
  ...
  return 0;
}
```

The main program is also a function

- it is called by the operation system at the start of the program
- it does not get any arguments (we will change this later)
- it returns with integer (`int`) value
  - Traditionally, if execution was OK, it gives 0-t, otherwise an error code

```
Process returned 0 (0x0)   execution time:  0.001 s
press ENTER to continue.
```

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

register: ??

Mechanism of function call


```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

register: ??

The main function allocates space for its local variables in the stack.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

```
T 0x1FF8: ????
b 0x1FFC: ????
a 0x2000: ????

register: ??
```

The main function allocates space for its local variables in the stack.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	????
b 0x1FFC:	????
a 0x2000:	????

register:	??

It gives values to its local variables.

Mechanism of function call


```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;                /* base */
  b = 3;             /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	????
b 0x1FFC:	????
a 0x2000:	2
register:	??

It gives values to its local variables.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	????
b 0x1FFC:	????
a 0x2000:	2

register:	??

It gives values to its local variables.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	????
b 0x1FFC:	3
a 0x2000:	2
register:	??

It gives values to its local variables.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;          /* base */
  b = 3;        /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

Function call: the main function creates a copy of variables b and a in the stack.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

Function call: the `main` function creates a copy of variables `b` and `a` in the stack.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	0x1FF0:	2
	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

Function call: the main function creates a copy of variables b and a in the stack.

Mechanism of function call


```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;           /* base */
  b = 3;         /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	Address	Value
	0x1FF0:	2
	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

Function call: the main function places the return address in the stack.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;           /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	0x1FEC:	15
	0x1FF0:	2
	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register:	??

Function call: the main function places the return address in the stack.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	Address	Value
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

The control is handed over to the `area` function, who will see the actual parameters as `x` and `y`

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;               /* base */
  b = 3;           /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	Address	Value
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

The area function allocates space for variable S in the stack

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	????
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

The area function allocates space for variable S in the stack

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	????
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

It calculates the value of S

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;          /* base */
  b = 3;        /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	6
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

It calculates the value of S

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;             /* base */
  b = 3;           /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	6
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: ??

It returns the value of S through a register.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	6
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: 6

It returns the value of S through a register.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;           /* base */
  b = 3;         /* height */
  T = area(a, b); /* area */
  return 0;
}
```

S	0x1FE8:	6
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: 6

It removes S from the stack.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;             /* base */
  b = 3;           /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	Address	Value
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: 6

It removes S from the stack.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;         /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	Address	Value
	0x1FEC:	15
x	0x1FF0:	2
y	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: 6

The control is given back to calling program segment, to the line that was saved.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;             /* base */
  b = 3;           /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	0x1FEC:	15
	0x1FF0:	2
	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register: 6

The control is given back to calling program segment, to the line that was saved.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;         /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	Address	Value
	0x1FEC:	15
	0x1FF0:	2
	0x1FF4:	3
T	0x1FF8:	????
b	0x1FFC:	3
a	0x2000:	2

register:	6

The main function copies the return value from the register.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;          /* base */
  b = 3;        /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	Address	Value
	0x1FEC:	15
	0x1FF0:	2
	0x1FF4:	3
T	0x1FF8:	6
b	0x1FFC:	3
a	0x2000:	2

register: 6

The `main` function copies the return value from the register.

Mechanism of function call


```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

	Address	Value
	0x1FEC:	15
	0x1FF0:	2
	0x1FF4:	3
T	0x1FF8:	6
b	0x1FFC:	3
a	0x2000:	2

register:	6

The `main` function removes the return address and the parameters from the stack.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;              /* base */
  b = 3;            /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	6
b 0x1FFC:	3
a 0x2000:	2

register:	6

The function call is finished.

Mechanism of function call


```
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	6
b 0x1FFC:	3
a 0x2000:	2

register:	6

The main function copies return value 0 into the register.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;             /* base */
  b = 3;           /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	6
b 0x1FFC:	3
a 0x2000:	2
register:	0

The main function copies return value 0 into the register.

Mechanism of function call

```c
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;            /* base */
  b = 3;          /* height */
  T = area(a, b); /* area */
  return 0;
}
```

T 0x1FF8:	6
b 0x1FFC:	3
a 0x2000:	2
register:	0

The `main` function removes its variables from the stack, and hands over the control to the operating system.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
/* Area of a rectangle */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Main program */
int main(void)
{
  int a, b, T;
  a = 2;          /* base */
  b = 3;        /* height */
  T = area(a, b); /* area */
  return 0;
}
```

register: 0

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Passing parameters by value

- Functions receive the value of the actual argument expressions.

Mechanism of function call

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Passing parameters by value

- Functions receive the value of the actual argument expressions.
- Parameters can be used as variables, that have an initial value assigned at the point of calling.

Mechanism of function call

Passing parameters by value

- Functions receive the value of the actual argument expressions.
- Parameters can be used as variables, that have an initial value assigned at the point of calling.
- Functions may modify the values of the parameters, but this has no effect on the calling program segment.

Visibility and life-cycle of variables

Local variables

1. parameters of functions

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

Visibility and life-cycle of variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Local variables

1. parameters of functions
2. variables declared inside a function

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

Visibility and life-cycle of variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Local variables

1. parameters of functions
2. variables declared inside a function

- They are created when entering into the function, and are erased when returning from the function.

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

Visibility and life-cycle of variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Local variables

1. parameters of functions
2. variables declared inside a function

- They are created when entering into the function, and are erased when returning from the function.
- They are invisible for program segments outside of the function. (also for the calling segment!)

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

Visibility and life-cycle of variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Local variables

1. parameters of functions
2. variables declared inside a function

- They are created when entering into the function, and are erased when returning from the function.
- They are invisible for program segments outside of the function. (also for the calling segment!)

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

- They exist throughout the life-cycle of the program.

Visibility and life-cycle of variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Local variables

1. parameters of functions
2. variables declared inside a function

- They are created when entering into the function, and are erased when returning from the function.
- They are invisible for program segments outside of the function. (also for the calling segment!)

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

- They exist throughout the life-cycle of the program.
- They are visible for everyone and can be modified by anyone!

Visibility and life-cycle of variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Local variables

1. parameters of functions
2. variables declared inside a function

- They are created when entering into the function, and are erased when returning from the function.
- They are invisible for program segments outside of the function. (also for the calling segment!)

Global variables – only for emergency cases!

Variables declared outside of functions (even outside of `main()`)

- They exist throughout the life-cycle of the program.
- They are visible for everyone and can be modified by anyone!
- In case of conflicts, the local variable masks out the global one.

Riddle

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

What will the following program print on the screen?

```
#include <stdio.h>

int a, b;

void func(int a)
{
  a = 2;
  b = 3;
}

int main(void)
{
  a = 1;
  func(a);
  printf("a: %d, b: %d\n", a, b);
  return 0;
}
```

link

A complex task

Let's create a C program, that asks two integer numbers from the user (`low` < `high`), and lists all prime numbers between these two numbers..

A complex task

Let's create a C program, that asks two integer numbers from the user (`low` < `high`), and lists all prime numbers between these two numbers..

- Pseudo-code of the solution broken into segments:

mainprogram

```
IN: low, high
FOR EACH i
  between low and high
  IF primetest(i) TRUE
    OUT: i
```

primetest(p)

```
FOR EACH i
  between 2 and root of p
  IF i divides p
    return FALSE
return TRUE
```

A complex task

Let's create a C program, that asks two integer numbers from the user (`low` < `high`), and lists all prime numbers between these two numbers..

- Pseudo-code of the solution broken into segments:

mainprogram

```
IN: low, high
FOR EACH i
  between low and high
  IF primetest(i) TRUE
    OUT: i
```

primetest(p)

```
FOR EACH i
  between 2 and root of p
  IF i divides p
    return FALSE
return TRUE
```

- Notice the role of the two *i* and *p*

Complex task – solution

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
#include <stdio.h> /* scanf, printf */

int low, high; /* global variables */

void read(void) /* inputting function */
{
  printf("Give a small and a larger number!\n");
  scanf("%d%d", &low, &high);
}

int isprime(int p) /* primetest function. */
{
  int i;
  for (i=2; i*i<=p; i=i+1) /* i from 2 to root of p */
    /* if p is dividable by i, not a prime */
    if (p%i == 0)
      return 0;
  return 1; /* if we get here, it is a prime */
}
```

Complex task – solution

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c

int main()
{
  int i;

  read(); /* we read the limits with a function */

  printf("Primes between %d and %d:\n", low, high);
  for (i=low; i<=high; i=i+1)
  {
    if (isprime(i)) /* we test with a function */
      printf("%d\n", i);
  }

  return 0;
}
```

link

Design principles

- Functions and programs communicate via parameters and return values.
- Except when this is their special task, functions
  - do not print on the screen,
  - do not read from keyboard,
  - do not use global variables.

Chapter 2

Arithmetic types of C

Types – Introduction

Type is

- Set of values
- Operations

Types – Introduction

Type is

- Set of values
- Operations

In a real computer – the set of values is limited

- We can not represent arbitrary large numbers

Types – Introduction

Type is

- Set of values
- Operations

In a real computer – the set of values is limited

- We can not represent arbitrary large numbers
- We can not represent numbers with arbitrary accuracy
  $\pi \neq 3.141592654$

Types – Introduction

Type is

- Set of values
- Operations
- Representation

In a real computer – the set of values is limited

- We can not represent arbitrary large numbers
- We can not represent numbers with arbitrary accuracy
  $\pi \neq 3.141592654$
- We must know the limits of what can be represented, in order to store our data

Types – Introduction

Type is

- Set of values
- Operations
- Representation

In a real computer – the set of values is limited

- We can not represent arbitrary large numbers
- We can not represent numbers with arbitrary accuracy
  $\pi \neq 3.141592654$
- We must know the limits of what can be represented, in order to store our data
  - without any loss of information or

Types – Introduction

Type is

- Set of values
- Operations
- Representation

In a real computer – the set of values is limited

- We can not represent arbitrary large numbers
- We can not represent numbers with arbitrary accuracy
  $\pi \neq 3.141592654$
- We must know the limits of what can be represented, in order to store our data
  - without any loss of information or
  - with an acceptable level of information loss, without wasting memory

Types of C language

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- void
- scalar
  - arithmetic
    - integer: integer, character, enumerated
    - floating-point
  - pointer
- function
- union
- compound
  - array
  - structure

Types of C language

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

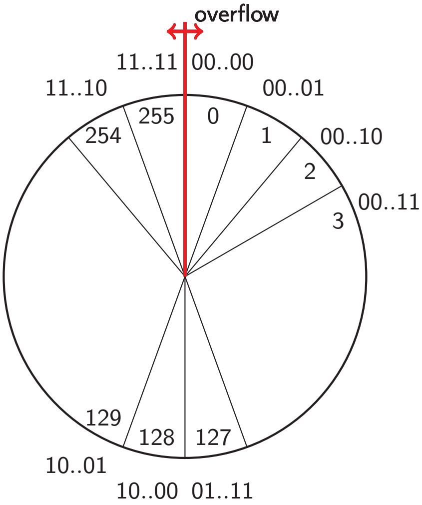
- void
- scalar
  - arithmetic
    - integer: integer, character, enumerated
    - floating-point
  - pointer
- function
- union
- compound
  - array
  - structure
- Today we will learn about them

Binary representation of integers

- Binary representation of unsigned integers stored in 8 bits

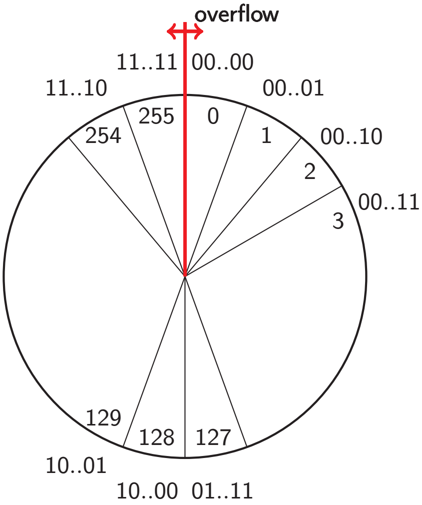
dec	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	hex
0	0	0	0	0	0	0	0	0	0x00
1	0	0	0	0	0	0	0	1	0x01
2	0	0	0	0	0	0	1	0	0x02
3	0	0	0	0	0	0	1	1	0x03
⋮	⋮							⋮	⋮
127	0	1	1	1	1	1	1	1	0x7F
128	1	0	0	0	0	0	0	0	0x80
129	1	0	0	0	0	0	0	1	0x81
⋮	⋮							⋮	⋮
253	1	1	1	1	1	1	0	1	0xFD
254	1	1	1	1	1	1	1	0	0xFE
255	1	1	1	1	1	1	1	1	0xFF

The overflow



- In case of unsigned integers stored in 8 bits

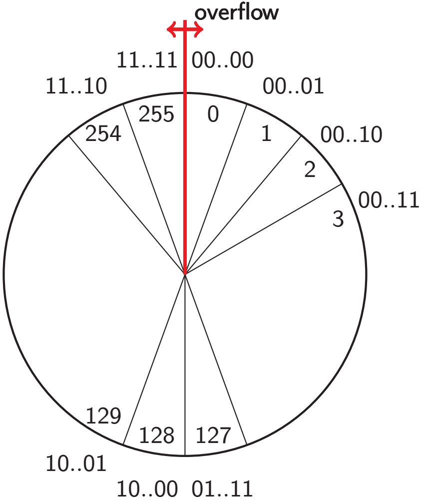
The overflow



- In case of unsigned integers stored in 8 bits
  - 255+1 = 0

The overflow

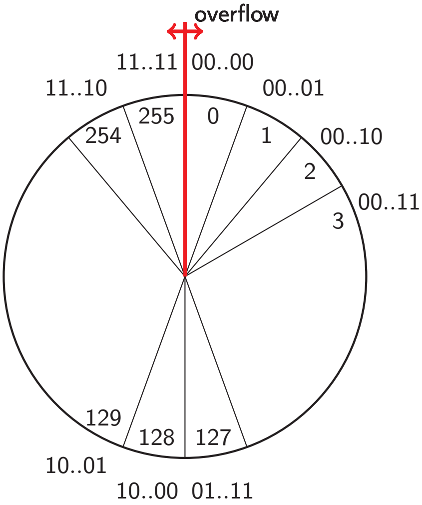
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- In case of unsigned integers stored in 8 bits
  - 255+1 = 0
  - 255+2 = 1

The overflow

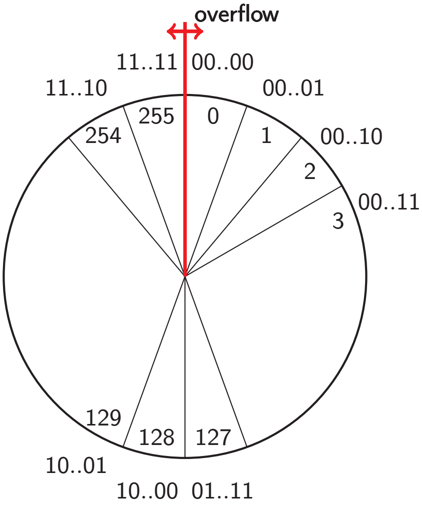
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- In case of unsigned integers stored in 8 bits
  - 255+1 = 0
  - 255+2 = 1
  - 2–3 = 255

The overflow

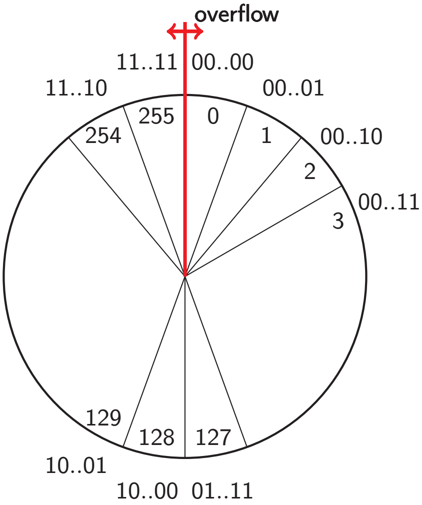
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- In case of unsigned integers stored in 8 bits
  - 255+1 = 0
  - 255+2 = 1
  - 2–3 = 255
- "modulo 256 arithmetic"

The overflow

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- In case of unsigned integers stored in 8 bits
  - 255+1 = 0
  - 255+2 = 1
  - 2–3 = 255
- ”modulo 256 arithmetic”
  - We always see the remainder of the result divided by 256

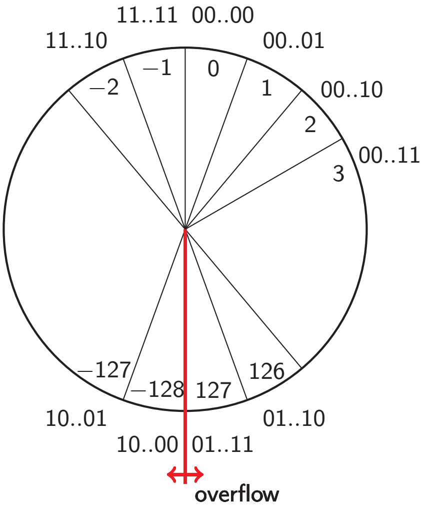
Two's complement representation of integers

- Two's complement representation of signed integers stored in 8 bits

dec	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	hex
0	0	0	0	0	0	0	0	0	0x00
1	0	0	0	0	0	0	0	1	0x01
2	0	0	0	0	0	0	1	0	0x02
3	0	0	0	0	0	0	1	1	0x03
⋮	⋮							⋮	⋮
127	0	1	1	1	1	1	1	1	0x7F
−128	1	0	0	0	0	0	0	0	0x80
−127	1	0	0	0	0	0	0	1	0x81
⋮	⋮							⋮	⋮
−3	1	1	1	1	1	1	0	1	0xFD
−2	1	1	1	1	1	1	1	0	0xFE
−1	1	1	1	1	1	1	1	1	0xFF

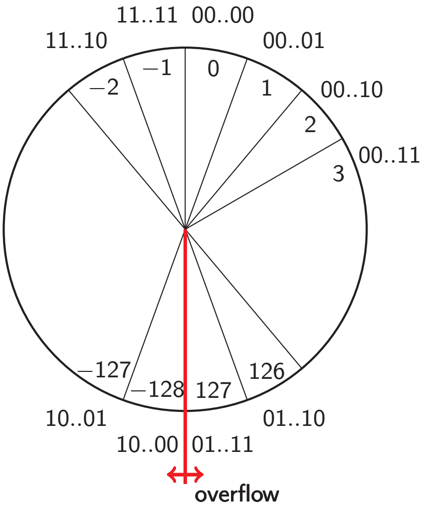
The overflow

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- In case of signed integers stored in 8 bits

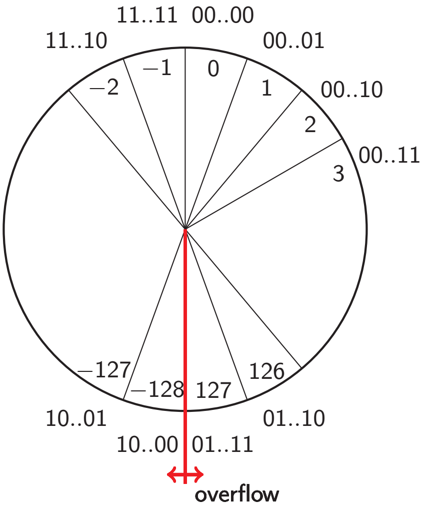
The overflow



- In case of signed integers stored in 8 bits
  - 127+1 = -128

The overflow

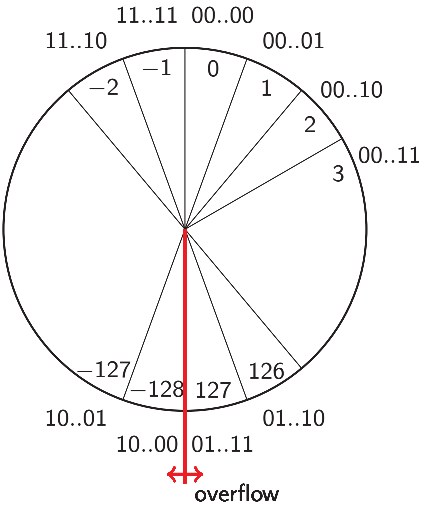
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- In case of signed integers stored in 8 bits
  - 127+1 = -128
  - 127+2 = -127

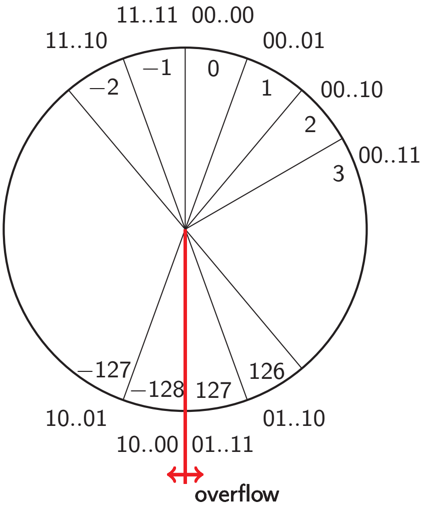
The overflow

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



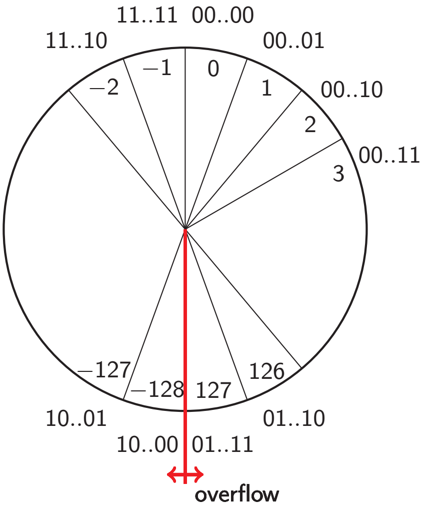
- In case of signed integers stored in 8 bits
  - 127+1 = -128
  - 127+2 = -127
  - -127-3 = 126

The overflow



- In case of signed integers stored in 8 bits
  - 127+1 = -128
  - 127+2 = -127
  - -127-3 = 126
- on the other hand

The overflow



- In case of signed integers stored in 8 bits
  - 127+1 = -128
  - 127+2 = -127
  - -127-3 = 126
- on the other hand
  - 2-3 = -1

Integer types in C

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

type	bit[1]	<limits.h>		printf
signed char	8	CHAR_MIN	CHAR_MAX	%hhd[2]
unsigned char	8	0	UCHAR_MAX	%hhu[2]
signed short int	16	SHRT_MIN	SHRT_MAX	%hd
unsigned short int	16	0	USHRT_MAX	%hu
signed int	32	INT_MIN	INT_MAX	%d
unsinged int	32	0	UINT_MAX	%u
signed long int	32	LONG_MIN	LONG_MAX	%ld
unsigned long int	32	0	ULONG_MAX	%lu
signed long long int[2]	64	LLONG_MIN	LLONG_MAX	%lld
unsigned long long int[2]	64	0	ULLONG_MAX	%llu

[1] Typical values, the standard only determines the minimum

[2] since the C99 standard

Declaration of integers

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Defaults
  - The `signed` sign-specifier can be omitted

```
int i;          /* signed int */
long int l;     /* signed long int */
```

Declaration of integers

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Defaults
  - The `signed` sign-specifier can be omitted

```
int i;              /* signed int */
long int l;         /* signed long int */
```

  - If there is sign- or length-modifier, the `int` can be omitted.

```
unsigned u;         /* unsigned int */
short s;            /* signed short int */
```

Integer types

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- An example on how to use the previous table: a program that runs for a very long time[3]

```
#include <limits.h> /* for integer limits */
#include <stdio.h>  /* for printf */

int main(void)
{ /* almost all long long int */
  long long i;

  for (i = LLONG_MIN; i < LLONG_MAX; i = i+1)
    printf("%lld\n", i);

  return 0;
}
```

link

[3]provided that `long long int` is 64 bit long, the program runs for 585 000 years if the computer prints 1 million numbers per second

Integer constants

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Specifying integer constants

```
int i1=0, i2=123, i4=-33;               /* decimal */
int o1=012, o2=01234567;                /* octal */
int h1=0x1a, h2=0x7fff, h3=0xAa1B       /* hexadecimal */

long l1=0x1al, l2=-33L;                 /* l or L */

unsigned u1=33u, u2=45U;                /* u or U */
unsigned long ul1=33uL, ul2=123lU; /* l and u */
```

Integer constants

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Specifying integer constants

```
int i1=0, i2=123, i4=-33;            /* decimal */
int o1=012, o2=01234567;             /* octal */
int h1=0x1a, h2=0x7fff, h3=0xAa1B  /* hexadecimal */

long l1=0x1al, l2=-33L;              /* l or L */

unsigned u1=33u, u2=45U;             /* u or U */
unsigned long ul1=33uL, ul2=123lU; /* l and u */
```

- If neither u or l is specified, the first type that is big enough is taken:
  1. `int`
  2. `unsigned int` – in case of hexa and octal constants
  3. `long`
  4. `unsigned long`

Why do we need to know the limits of number representations?

Let's determine the following value!

$$\binom{15}{12} = \frac{15!}{12! \cdot (15-12)!}$$

(What is the number of possibilities of selecting 12 out of 15 different chocolates?)

Why do we need to know the limits of number representations?

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's determine the following value!

$$\binom{15}{12} = \frac{15!}{12! \cdot (15-12)!}$$

(What is the number of possibilities of selecting 12 out of 15 different chocolates?)

- The value of the numerator is $15! = 1\ 307\ 674\ 368\ 000$

Why do we need to know the limits of number representations?

Let's determine the following value!

$$\binom{15}{12} = \frac{15!}{12! \cdot (15-12)!}$$

(What is the number of possibilities of selecting 12 out of 15 different chocolates?)

- The value of the numerator is $15! = 1\ 307\ 674\ 368\ 000$
- The value of the denominator is $12! \cdot 3! = 2\ 874\ 009\ 600$

Why do we need to know the limits of number representations?

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's determine the following value!

$$\binom{15}{12} = \frac{15!}{12! \cdot (15-12)!}$$

(What is the number of possibilities of selecting 12 out of 15 different chocolates?)

- The value of the numerator is $15! = 1\ 307\ 674\ 368\ 000$
- The value of the denominator is $12! \cdot 3! = 2\ 874\ 009\ 600$
- None of them can be represented as a 32 bits `int`!

Why do we need to know the limits of number representations?

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's determine the following value!

$$\binom{15}{12} = \frac{15!}{12! \cdot (15-12)!}$$

(What is the number of possibilities of selecting 12 out of 15 different chocolates?)

- The value of the numerator is $15! = 1\,307\,674\,368\,000$
- The value of the denominator is $12! \cdot 3! = 2\,874\,009\,600$
- None of them can be represented as a 32 bits `int`!
- But with simplifying the expression

$$\frac{15 \cdot 14 \cdot 13}{3 \cdot 2 \cdot 1} = \frac{2730}{6} = 455$$

  all parts can be calculated without any problem, even on 12 bits.

Representing characters – The ASCII table

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- 128 characters, that can be indexed with numbers 0x00–0x7f

Code	00	10	20	30	40	50	60	70
+00	NUL	DLE	␣	0	@	P	`	p
+01	SOH	DC1	!	1	A	Q	a	q
+02	STX	DC2	"	2	B	R	b	r
+03	ETX	DC3	#	3	C	S	c	s
+04	EOT	DC4	$	4	D	T	d	t
+05	ENQ	NAK	%	5	E	U	e	u
+06	ACK	SYN	&	6	F	V	f	v
+07	BEL	ETB	'	7	G	W	g	w
+08	BS	CAN	(	8	H	X	h	x
+09	HT	EM	)	9	I	Y	i	y
+0a	LF	SUB	*	:	J	Z	j	z
+0b	VT	ESC	+	;	K	[	k	{
+0c	FF	FS	,	<	L	\	l	|
+0d	CR	GS	-	=	M	]	m	}
+0e	SO	RS	.	>	N	^	n	~
+0f	SI	US	/	?	O	_	o	DEL

Storing, printing and reading characters

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Characters (indexes of the ASCII table) are stored in `char` type
- Printing of the elements of the ASCII table is done with `%c` format code.

```
char ch = 0x61; /* hex 61 = dec 97 */
printf("%d: %c\n", ch, ch);
ch = ch+1; /* its value will be hex 62 = dec 98  */
printf("%d: %c\n", ch, ch);
```

Storing, printing and reading characters

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Characters (indexes of the ASCII table) are stored in `char` type
- Printing of the elements of the ASCII table is done with `%c` format code.

```c
char ch = 0x61; /* hex 61 = dec 97 */
printf("%d: %c\n", ch, ch);
ch = ch+1; /* its value will be hex 62 = dec 98  */
printf("%d: %c\n", ch, ch);
```

- Output of the program

```
97:  a
98:  b
```

Storing, printing and reading characters

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Characters (indexes of the ASCII table) are stored in `char` type
- Printing of the elements of the ASCII table is done with `%c` format code.

```
char ch = 0x61; /* hex 61 = dec 97 */
printf("%d: %c\n", ch, ch);
ch = ch+1; /* its value will be hex 62 = dec 98  */
printf("%d: %c\n", ch, ch);
```

- Output of the program

```
97:  a
98:  b
```

- Does it mean we have to learn the ASCII-codes to be able to print characters?

Character constants

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- A character placed between apostrophes is equivalent to its ASCII-code

```
char ch = 'a'; /* 0x61 ASCII-code is copied to ch */
printf("%d: %c\n", ch, ch);
ch = ch+1;
printf("%d: %c\n", ch, ch);
```

Character constants

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- A character placed between apostrophes is equivalent to its ASCII-code

```
char ch = 'a'; /* 0x61 ASCII-code is copied to ch */
printf("%d: %c\n", ch, ch);
ch = ch+1;
printf("%d: %c\n", ch, ch);
```

```
97: a
98: b
```

Character constants

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- A character placed between apostrophes is equivalent to its ASCII-code

```c
char ch = 'a'; /* 0x61 ASCII-code is copied to ch */
printf("%d: %c\n", ch, ch);
ch = ch+1;
printf("%d: %c\n", ch, ch);
```

```
97: a
98: b
```

- Beware! `'0'` $\neq 0$!

```c
char n = '0'; /* 0x30 ASCII-code is copied to ch !!! */
printf("%d: %c\n", n, n);
```

Character constants

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- A character placed between apostrophes is equivalent to its ASCII-code

```
char ch = 'a'; /* 0x61 ASCII-code is copied to ch */
printf("%d: %c\n", ch, ch);
ch = ch+1;
printf("%d: %c\n", ch, ch);
```

```
97: a
98: b
```

- Beware! `'0'` $\neq 0$!

```
char n = '0'; /* 0x30 ASCII-code is copied to ch !!! */
printf("%d: %c\n", n, n);
```

```
48: 0
```

Character constants

- Special character constants – that would be hard to type. . .

0x00	`\0`	null character (NUL)
0x07	`\a`	bell (BEL)
0x08	`\b`	backspace (BS)
0x09	`\t`	tabulator (HT)
0x0a	`\n`	line feed (LF)
0x0b	`\v`	vertical tab (VT)
0x0c	`\f`	form feed (FF)
0x0d	`\r`	carriage return (CR)
0x22	`\''`	quotation mark
0x27	`\'`	apostrophe
0x5c	`\\`	backslash

Character or integer number?

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- In C language characters are equivalent to integer numbers

Character or integer number?

- In C language characters are equivalent to integer numbers
- It will be decided only at the moment of displaying how an integer value is printed: as a number or as a character (`%d` or `%c`)

Character or integer number?

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- In C language characters are equivalent to integer numbers
- It will be decided only at the moment of displaying how an integer value is printed: as a number or as a character (`%d` or `%c`)
- We can perform the same operations on characters as on integers (adding, subtracting, etc. . .)

Character or integer number?

- In C language characters are equivalent to integer numbers
- It will be decided only at the moment of displaying how an integer value is printed: as a number or as a character (`%d` or `%c`)
- We can perform the same operations on characters as on integers (adding, subtracting, etc. . .)
- But what is the point in adding-subtracting characters?

Operations with characters

Let's write a program, that reads characters as long as a new line character has not arrived. After this the program should print out the sum of the read (scanned) digits.

Operations with characters

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's write a program, that reads characters as long as a new line character has not arrived. After this the program should print out the sum of the read (scanned) digits.

```c
char c;
int sum = 0;
do
{
  scanf("%c", &c);                /* reading */
  if (c >= '0' && c <= '9')       /* if numerical digit */
    sum = sum + (c-'0');          /* summing */
}
while (c != '\n');                /* stop condition */
printf("The sum is: %d\n", sum);
```

Operations with characters

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's write a program, that reads characters as long as a new line character has not arrived. After this the program should print out the sum of the read (scanned) digits.

```
char c;
int sum = 0;
do
{
  scanf("%c", &c);              /* reading */
  if (c >= '0' && c <= '9')     /* if numerical digit */
    sum = sum + (c-'0');        /* summing */
}
while (c != '\n');              /* stop condition */
printf("The sum is: %d\n", sum);
```

```
The airplane has landed at 12:35 this afternoon
The sum is:  11
```

Operations with characters

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's write a function, that converts the lowercase letters of the English alphabet to uppercase, but leaves all other characters unchanged.

Operations with characters

Let's write a function, that converts the lowercase letters of the English alphabet to uppercase, but leaves all other characters unchanged.

```
char toupper(char c)
{
  if (c >= 'a' && c <= 'z') /* if lowercase */
  {
    return c - 'a' + 'A';
  }
  return c;
}
```

Floating-point types

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Normal form

$$23.2457 = (-1)^{0} \cdot 2.3245700 \cdot 10^{+001}$$

$$-0.001822326 = (-1)^{1} \cdot 1.8223260 \cdot 10^{-003}$$

Floating-point types

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Normal form

$$23.2457 = (-1)^{0} \cdot 2.3245700 \cdot 10^{+001}$$
$$-0.001822326 = (-1)^{1} \cdot 1.8223260 \cdot 10^{-003}$$

Representation of the normal form

- Floating-point fractional = sign bit + mantissa + exponent

Floating-point types

- Normal form

$$23.2457 = (-1)^{0} \cdot 2.3245700 \cdot 10^{+001}$$
$$-0.001822326 = (-1)^{1} \cdot 1.8223260 \cdot 10^{-003}$$

Representation of the normal form

- Floating-point fractional = sign bit + mantissa + exponent
  1. sign bit: 0–positive, 1–negative

Floating-point types

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Normal form

$$23.2457 = (-1)^0 \cdot 2.3245700 \cdot 10^{+001}$$
$$-0.001822326 = (-1)^1 \cdot 1.8223260 \cdot 10^{-003}$$

Representation of the normal form

- Floating-point fractional = sign bit + mantissa + exponent
  1. sign bit: 0–positive, 1–negative
  2. mantissa: unsigned integer (without the decimal comma), because of normalization, the first digit is ≥ 1

Floating-point types

- Normal form

$$23.2457 = (-1)^{0} \cdot 2.3245700 \cdot 10^{+001}$$
$$-0.001822326 = (-1)^{1} \cdot 1.8223260 \cdot 10^{-003}$$

Representation of the normal form

- Floating-point fractional = sign bit + mantissa + exponent
  1. sign bit: 0–positive, 1–negative
  2. mantissa: unsigned integer (without the decimal comma), because of normalization, the first digit is ≥ 1
  3. exponent (or order, characteristic): signed integer

Floating-point types

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Binary normal form

$$5.0 = 1.25 \cdot 4 = (-1)^{0} \cdot 1.0100_{\mathrm{b}} \cdot 2^{010_{\mathrm{b}}}$$

0	**0100**	**010**

[4] the leading bit is implicit

Floating-point types

- Binary normal form

$$5.0 = 1.25 \cdot 4 = (-1)^{0} \cdot 1.0100_{\mathrm{b}} \cdot 2^{010_{\mathrm{b}}}$$

0	**0100**	**010**

Representation of binary normal form

- Floating-point fractional = sign bit + mantissa + exponent

[4]the leading bit is implicit

Floating-point types

- Binary normal form

$$5.0 = 1.25 \cdot 4 = (-1)^{0} \cdot 1.0100_{\mathrm{b}} \cdot 2^{010_{\mathrm{b}}}$$

0	0100	010

Representation of binary normal form

- Floating-point fractional = sign bit + mantissa + exponent
  1. sign bit: 0–positive, 1–negative

[4] the leading bit is implicit

Floating-point types

- Binary normal form

$$5.0 = 1.25 \cdot 4 = (-1)^{0} \cdot 1.0100_{\mathrm{b}} \cdot 2^{010_{\mathrm{b}}}$$

0	**0100**	**010**

Representation of binary normal form

- Floating-point fractional = sign bit + mantissa + exponent
  1. sign bit: 0–positive, 1–negative
  2. mantissa: unsigned integer (without the **binary comma**), because of normalization, the first digit is $= 1$, so we don't store it[4].

[4] the leading bit is implicit

Floating-point types

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Binary normal form

$$5.0 = 1.25 \cdot 4 = (-1)^0 \cdot 1.0100_b \cdot 2^{010_b}$$

0	**0100**	**010**

Representation of binary normal form

- Floating-point fractional = sign bit + mantissa + exponent
  1. sign bit: 0–positive, 1–negative
  2. mantissa: unsigned integer (without the **binary comma**), because of normalization, the first digit is = 1, so we don't store it[4].
  3. exponent: signed integer

[4] the leading bit is implicit

Floating-point types in C

- Floating-point types of C

	typical values			
type	bits	mantissa	exponent	printf/scanf
`float`	32 bits	23 bits	8 bits	`%f`
`double`	64 bits	52 bits	11 bits	`%f`/`%lf`
`long double`	128 bits	112 bits	15 bits	`%Lf`

Floating-point types in C

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Floating-point types of C

	typical values			
type	bits	mantissa	exponent	printf/scanf
`float`	32 bits	23 bits	8 bits	`%f`
`double`	64 bits	52 bits	11 bits	`%f`/`%lf`
`long double`	128 bits	112 bits	15 bits	`%Lf`

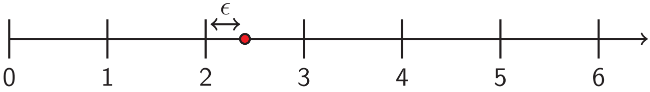
- Floating-point constants

```
float       f1=12.3f , f2=12.F , f3=.5f , f4=1.2e-3F ;
double      d1=12.3  , d2=12.  , d3=.5  , d4=1.2e-3  ;
long double l1=12.3l , l2=12.L , l3=.5l , l4=1.2e-3L ;
```

- In C we use decimal point and not a comma!

Representation accuracy of integer types

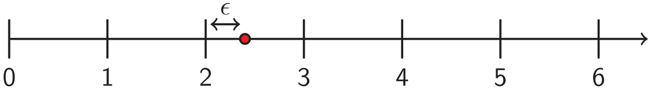
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



Absolute accuracy of number representation

It is the maximal ϵ error of representing an arbitrary real number with the closest integer

Representation accuracy of integer types

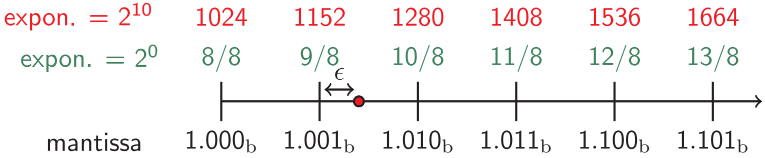


Absolute accuracy of number representation

It is the maximal ϵ error of representing an arbitrary real number with the closest integer

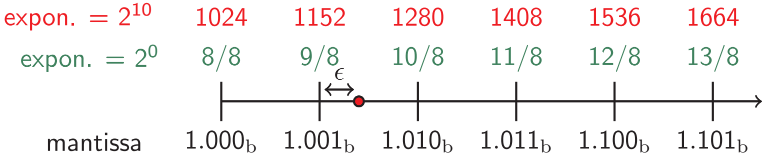
- The absolute accuracy of representing with integer types is 0.5

Representation accuracy of floating-point numbers



Representation accuracy of floating-point numbers

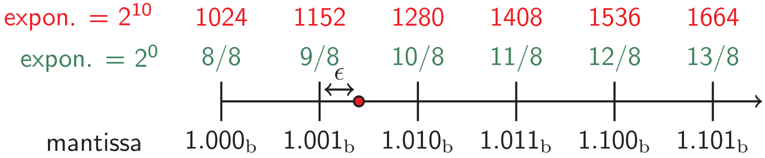
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- in this example
  - The (absolute) representation accuracy of the mantissa is 1/16

Representation accuracy of floating-point numbers

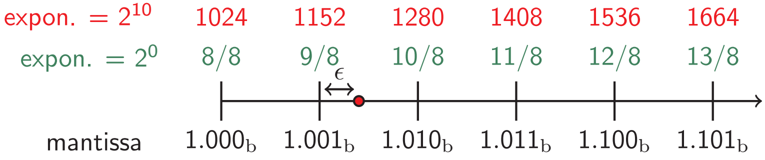
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- in this example
  - The (absolute) representation accuracy of the mantissa is $1/16$
  - If the exponent is 2^0, the representation accuracy is $1/16$

Representation accuracy of floating-point numbers

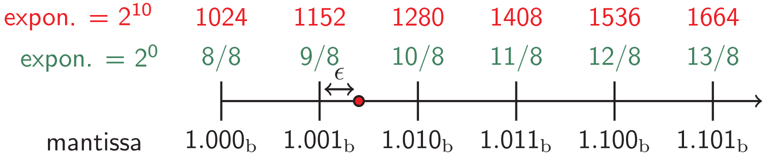
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- in this example
  - The (absolute) representation accuracy of the mantissa is $1/16$
  - If the exponent is 2^0, the representation accuracy is $1/16$
  - If the exponent is 2^{10}, the representation accuracy is $2^{10}/16 = 64$

Representation accuracy of floating-point numbers

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- in this example
  - The (absolute) representation accuracy of the mantissa is $1/16$
  - If the exponent is 2^0, the representation accuracy is $1/16$
  - If the exponent is 2^{10}, the representation accuracy is $2^{10}/16 = 64$
- There is no absolute, only relative accuracy, that is, in this present case, 3 bits.

Consequences of finite number representation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- As the floating-point number representation is not accurate, we must not check the equality of results of operations!

$$\frac{22}{7} + \frac{3}{7} \neq \frac{25}{7}$$

instead

$$\left|\frac{22}{7} + \frac{3}{7} - \frac{25}{7}\right| < \varepsilon$$

Consequences of finite number representation

- As the floating-point number representation is not accurate, we must not check the equality of results of operations!

$$\frac{22}{7} + \frac{3}{7} \neq \frac{25}{7}$$

instead

$$\left|\frac{22}{7} + \frac{3}{7} - \frac{25}{7}\right| < \varepsilon$$

- The exponent will magnify the rounding error of the finite long mantissa, thus the large numbers are much less accurate than small numbers. The errors of the large numbers can "eat up" the small ones:

$$A + a - A \neq a$$

Consequences of the binary representation of numbers

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- A decimal finite number might not be finite in binary form, eg.:

$$0{,}1_{\mathrm{d}} = 0{,}0\overline{0011}_{\mathrm{b}}$$

Consequences of the binary representation of numbers

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- A decimal finite number might not be finite in binary form, eg.:

$$0{,}1_{\mathrm{d}} = 0{,}0\overline{0011}_{\mathrm{b}}$$

- How many times will be this cycle repeated?

```
double d;
for (d = 0.0; d < 1.0; d = d+0.1) /* 10? 11? */
{
   ...
}
```

Consequences of the binary representation of numbers

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- A decimal finite number might not be finite in binary form, eg.:

$$0{,}1_{\mathrm{d}} = 0{,}0\overline{0011}_{\mathrm{b}}$$

- How many times will be this cycle repeated?

```
double d;
for (d = 0.0; d < 1.0; d = d+0.1) /* 10? 11? */
{
   ...
}
```

- The good solution is:

```
double d;
double eps = 1e-3; /* what is the right eps for here? */
for (d = 0.0; d < 1.0-eps; d = d+0.1) /* 10 times */
{
   ...
}
```

Thank you for your attention.