Programação Imperativa

LEI, 2021-22

Aula 1

Avaliação

- Época normal
  - 75% Teste (31 Maio)
  - 25% Mini-testes
    - Online: 7 Março, 21 Março, 18 Abril, 16 Maio (às 19h00 no BB)
    - Presenciais: 4 a 8 de Abril, 3 a 9 de Maio (nas aulas TP)
- Época de recurso
  - 100% Exame (21 Junho)

Docentes

- Teóricas
  - Alcino Cunha (T1), alcino@di.uminho.pt
  - José Bernardo Barros (T2), jbb@di.uminho.pt
- Teórico-práticas
  - Alcino Cunha
  - Catarina Machado
  - José Bernardo Barros
  - José Carlos Bacelar
  - Leandro Gomes
  - Lisandra Silva
  - Óscar Ribeiro

Haskell

- Linguagem de programação de uso genérico
- Funcional
- Estáticamente (fortemente) tipada
- Interpretada e compilada
- Alto nível
- Criada em 1990

C

- Linguagem de programação de uso genérico
- Imperativa e procedimental
- Estaticamente (fracamente) tipada
- Compilada
- Baixo nível
- Criada em 1972

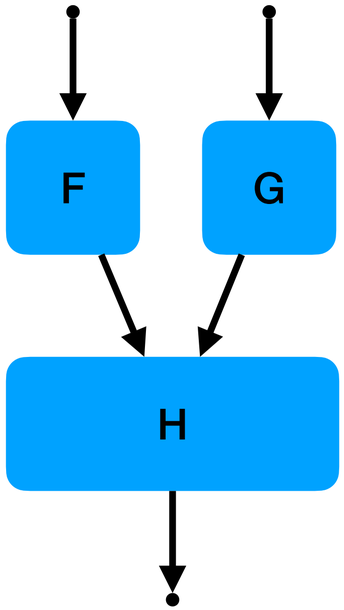
SECOND EDITION

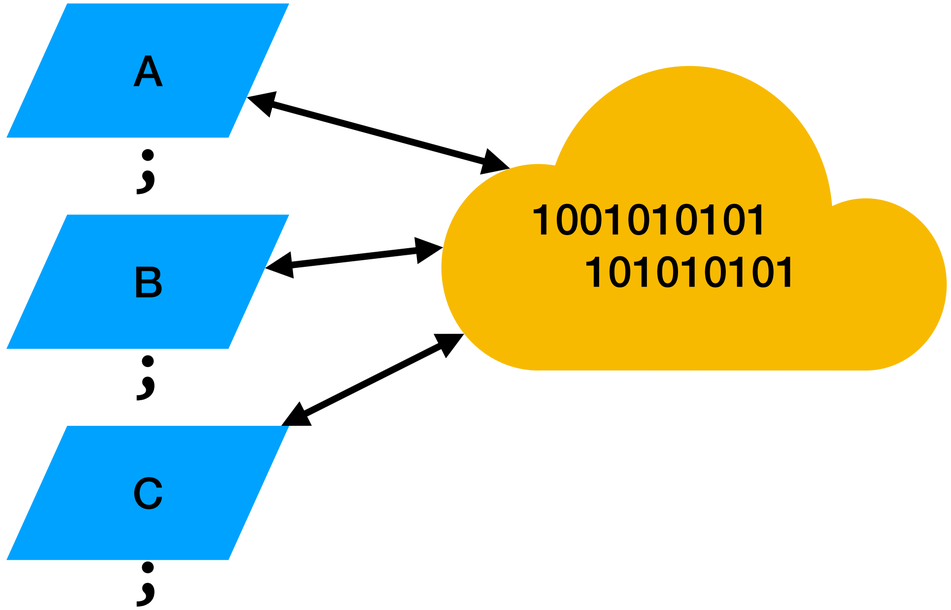
THE C PROGRAMMING LANGUAGE

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

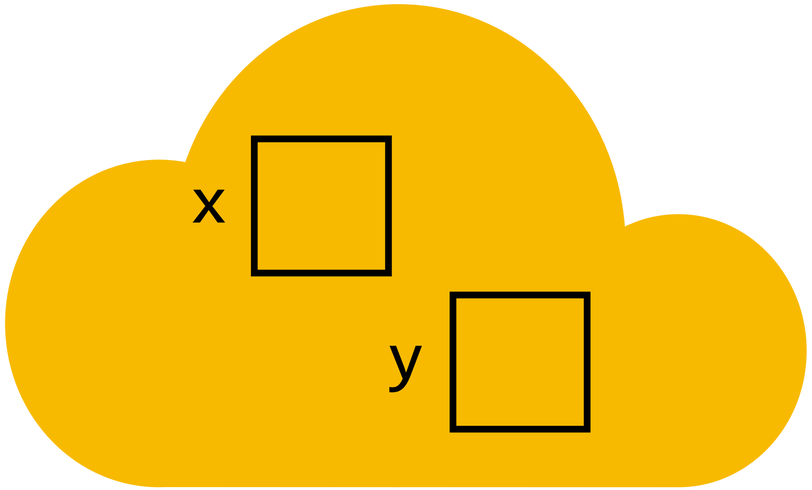
PRENTICE HALL SOFTWARE SERIES

Funcional vs Imperativa

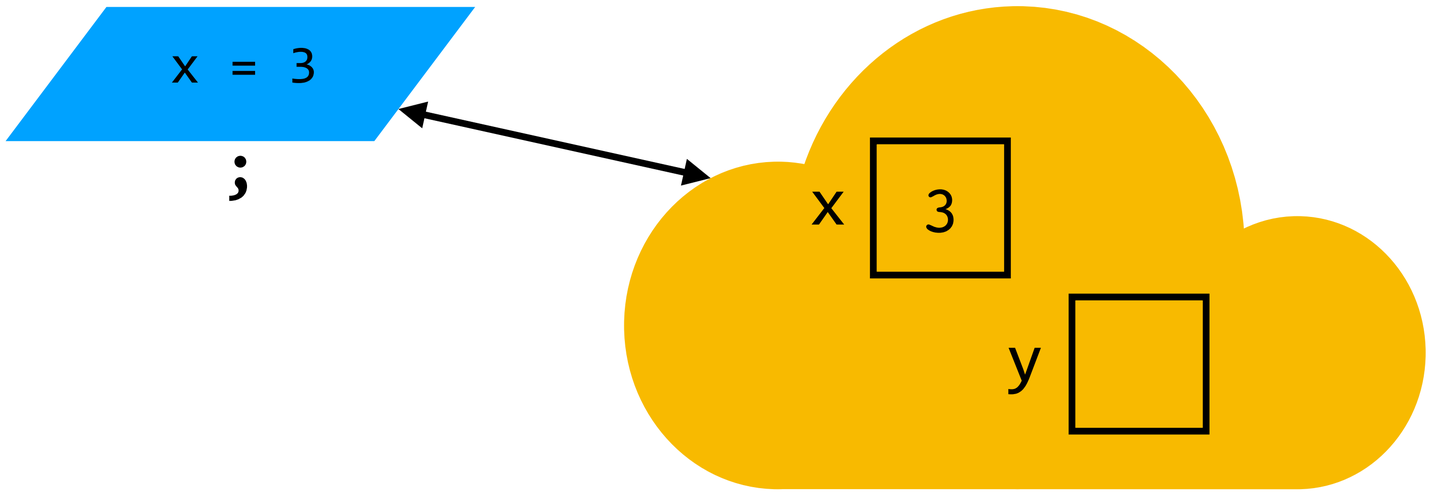




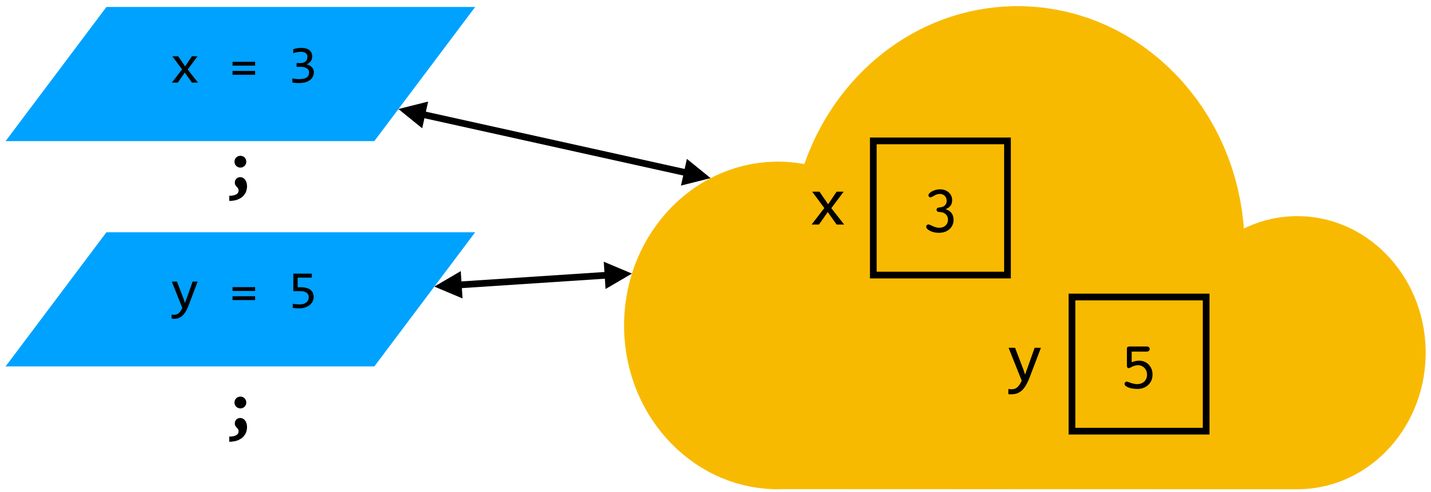
Variáveis e Atribuição



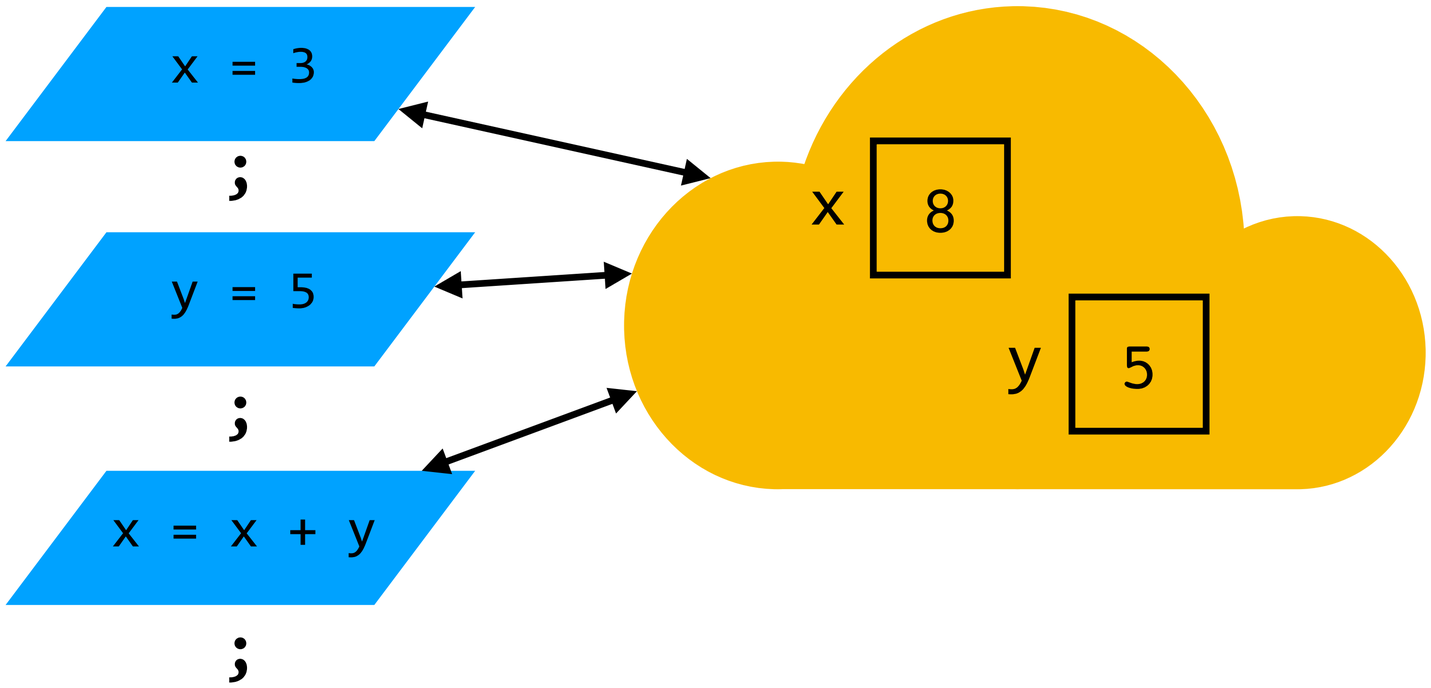
Variáveis e Atribuição



Variáveis e Atribuição



Variáveis e Atribuição



Programas

- Um *programa* C é uma sequência de declarações de *variáveis, tipos* ou *funções, directivas de pré-processamento*
- Uma variável, tipo ou função só pode ser usada se declarada antes
- O espaçamento e indentação não têm significado
- As declarações são terminadas por ponto e vírgula
- Um programa pode estar dividido em vários ficheiros (*aka* módulos ou bibliotecas)
- A execução de um programa começa na função especial `main`

Declaração de variáveis

tipo id;

tipo id = expr;

tipo id_0, id_1, …;

tipo id_0 = $expr_0$, id_1 = $expr_0$, …;

Tipos numéricos

Tipo	Modificadores	Tamanho
`char`	`signed`, `unsigned`	≥ 8
`short`	`signed` (default), `unsigned`	≥ 16
`int`	`signed` (default), `unsigned`	≥ 16
`long`	`signed` (default), `unsigned`	≥ 32
`float`		precisão simples
`double`		precisão dupla

Funções

- A *declaração* de uma função inclui os tipos dos parâmetros e o tipo do valor retornado
- Uma função pode não ter parâmetros, nem retornar qualquer valor, usando-se neste caso o tipo especial **`void`** como tipo de retorno
- A declaração de uma função é normalmente seguida da respectiva *definição*
- A definição de uma função é uma sequência declarações (de variáveis ou tipos) e de *comandos* entre chavetas
- Tal como as declarações, os comandos são terminados por ponto e vírgula
- O comando **`return`** é usado para retornar um valor

Definição de funções

```
tipo id (tipo1 id1, …, tipon idn) {

   // declarações e comandos

   return expr;
}
```

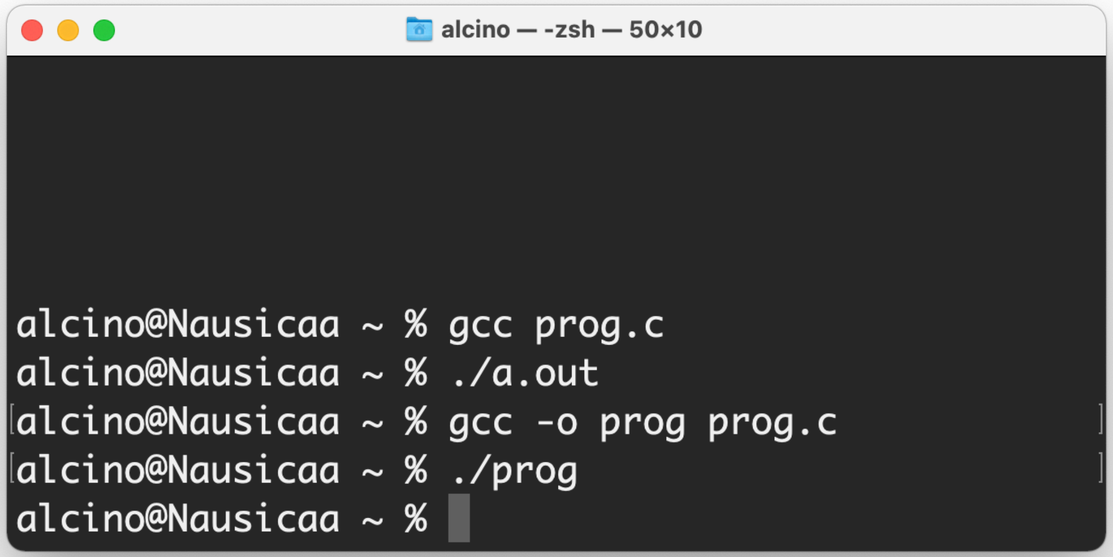
Exemplo

```
int dobro(int a) {
  int r;
  r = 2*a;
  return r;
}

int main() {
  int r;
  r = dobro(3);
  return 0;
}
```

Compilação

```c
int dobro(int a) {
   int r;
   r = 2*a;
   return r;
}

int main() {
   int r;
   r = dobro(3);
   return 0;
}
```

prog.c



Aula 2

Declaração vs Definição

```
int dobro(int);       // Declaração

int main() {
  int r;
  r = dobro(3);
  return 0;
}

int dobro(int a) {    // Definição
  int r;
  r = 2*a;
  return r;
}
```

Directiva #include

```c
int dobro(int a) {
  int r;
  r = 2*a;
  return r;
}
```

dobro.c

```c
#include "dobro.c"

int main() {
  int r;
  r = dobro(3);
  return 0;
}
```

prog.c

Bibliotecas pré-definidas

Nome	Conteúdo
`<stdio.h>`	Input e output
`<stdlib.h>`	Conversão de tipos, geração de números aleatórios, alocação de memória, …
`<math.h>`	Funções matemáticas (exponenciação, raíz quadrada, trigonométricas, …)
`<string.h>`	Manipulação de strings

A função `printf`

```c
#include <stdio.h>

int main() {
  printf("Olá mundo!\n");
  printf("O dobro de %s é %d.\n", "dois", 4);
  return 0;
}
```

Códigos de formatação

Código	Formatação
%d	Número inteiro em notação decimal com sinal
%x	Número inteiro em notação hexadecimal
%f	Número real em notação decimal
%e	Número real em notação científica
%c	Caracter
%s	String
%%	Caracter '%'

Códigos de formatação

Código	Formatação
`%`*n*`d`	Número inteiro em notação decimal com sinal, sendo *n* o número mínimo de caracteres a imprimir. Se o tamanho do número for menor são impressos espaços à esquerda.
`%.`*p*`f`	Número real em notação decimal, sendo *p* o número de dígitos à direita da vírgula
`%`*n*`.`*p*`f`	Idem, sendo *n* o número mínimo de caracteres a imprimir.

Comandos vs Expressões

- Os *comandos* afectam o estado do programa
  - Uma atribuição é um comando
- As *expressões* denotam um valor
  - Definidas à custa de constantes, operadores e invocação de funções

Expressões aritméticas

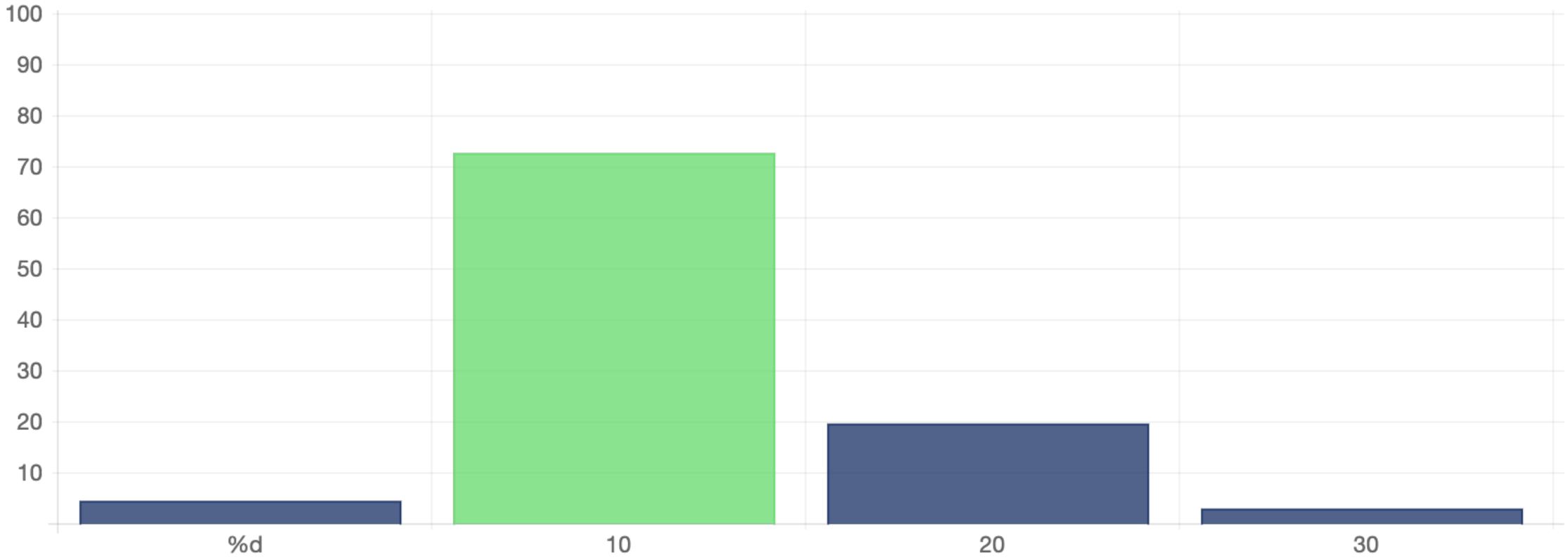
Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira


QUIZ!

Que imprime o seguinte programa?

```c
#include <stdio.h>

int main() {
  int x = 20, y = 10;
  x = x + y;
  y = x - y;
  x = x - y;
  printf("%d\n", x);
  return 0;
}
```

Que imprime o seguinte programa?

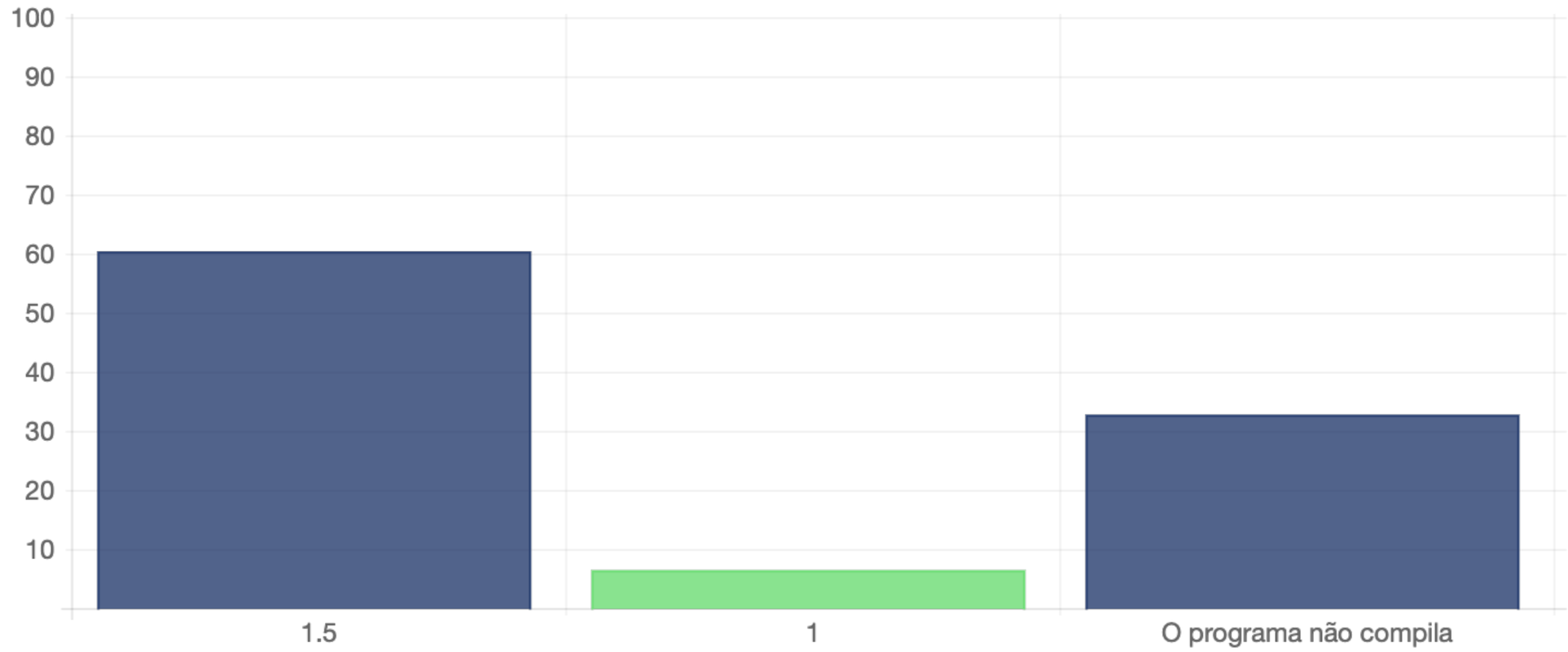


https://pythontutor.com/c.html

Qual o valor de `f` depois da atribuição?

```
int main() {
  float f;
  f = 3 / 2;
  return 0;
}
```

Qual o valor de `f` depois da atribuição?



Conversões entre tipos

- Numa operação aritmética o operando de tipo “mais pequeno” é convertido para o tipo “maior”
  - Se um operando é um **`double`** o outro é convertido para **`double`**
  - Senão, se um operando é um **`float`** o outro é convertido para **`float`**
  - Senão, qualquer **`char`** ou **`short`** é promovido para **`int`**
  - Depois, se um operando é **`long`** o outro é convertido para **`long`**
- A operação é depois realizada no tipo “maior”
- Numa atribuição numérica o valor da expressão é convertido implicitamente para o tipo da variável
- Uma conversão pode também ser feita de forma explícita com o operador unário (*`tipo`*)

Conversões entre tipos

Divisão exacta

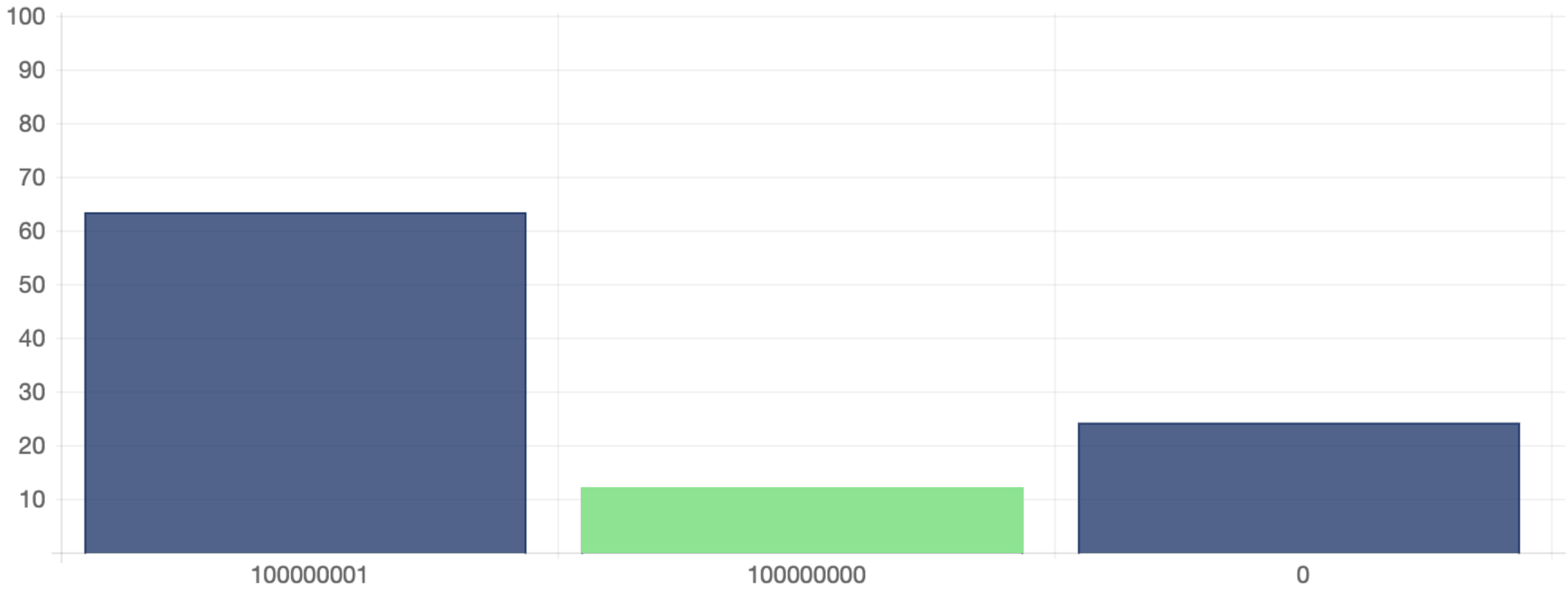
```c
int main() {
  float f;
  f = 3 / 2.0;
  return 0;
}
```

```c
int main() {
  float f;
  f = 3 / (float) 2;
  return 0;
}
```

Qual o valor de x no final?

```
int main() {
  int x = 100000001;
  float f;
  f = x;
  x = f;
  return 0;
};
```

Qual o valor de x no final?



Máximo

```c
#include <stdlib.h>

int max(int a, int b) {
  int r;
  r = (a + b + abs(a-b)) / 2;
  return r;
}
```

Condicional

if (*expr*) cmd_1; **else** cmd_2;

if (*expr*) *cmd*;

Operadores relacionais

Operador	Significado
==	Igualdade
!=	Diferença
<	Menor
<=	Menor ou igual
>	Maior
<=	Menor ou igual

Operadores lógicos

Operador	Significado
!	Negação
&&	Conjunção
||	Disjunção

Booleanos

- Não existe o tipo booleano em C
- Qualquer expressão numérica pode ser usada como booleano
  - O valor `0` corresponde a falso
  - Um valor diferente de `0` corresponde a verdadeiro
- Os operadores relacionais e lógicos devolvem `0` quando o resultado é falso e `1` quando é verdadeiro

Máximo

```
int max(int a, int b) {
  int r;
  if (a > b) r = a; else r = b;
  return r;
}
```

Máximo

```
int max(int a, int b) {
  if (a > b) return a;
  return b;
}
```

Aula 3

Blocos de comandos

- Em qualquer sítio onde é esperado um comando podemos colocar um *bloco de comandos*
- Um bloco de comandos é uma sequência de comandos ou declarações de variáveis entre chavetas
- As variáveis declaradas num bloco só podem ser usadas nesse bloco
- A definição de uma função é de facto um bloco de comandos

Máximo

```
int max(int a, int b) {
 if (a > b) {
    int r;
    r = a;
    return r;
  } else {
    int m;
    m = b;
    return m;
  }
}
```

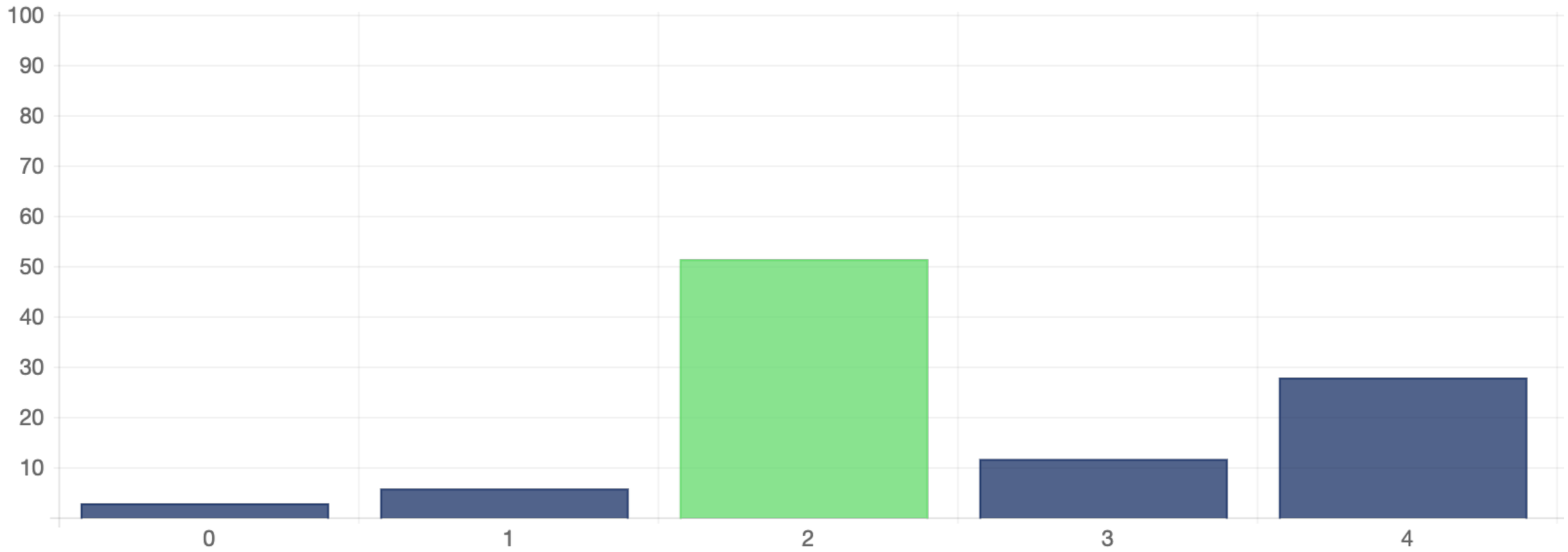
Expressões lógicas

- A avaliação de uma expressão lógica é feita da esquerda para a direita
- Termina logo que seja possível determinar o valor da expressão

Quantas comparações faz `isalpha('0')`?

```
int isalpha(int c) {
  return ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'));
}
```

Quantas comparações faz `isalpha('0')`?



Ciclo `while`

```
while (expr) cmd;
```

Factorial

```
int fact(int n) {
  int r = 1;
  int i = 1;
  while (i <= n) {
    r = r * i;
    i = i + 1;
  }
  return r;
}
```

Factorial

```
int fact(int n) {
  int r = 1;
  int i = 1;
  while (i <= n) {
    r = r * i;
    i = i + 1;
  }
  return r;
}
```

Factorial

```
int fact(int n) {
  int r = 1;
  int i = 1;
  while (i <= n) {
    r = r * i;
    i = i + 1;
  }
  return r;
}
```

n	r	i	i <= n

Factorial

```
int fact(int n) {
  int r = 1;
  int i = 1;
  while (i <= n) {
    r = r * i;
    i = i + 1;
  }
  return r;
}
```

n	r	i	i <= n
5	1	1	1

Factorial

```
int fact(int n) {
  int r = 1;
  int i = 1;
  while (i <= n) {
    r = r * i;
    i = i + 1;
  }
  return r;
}
```

n	r	i	i <= n
5	1	1	1
5	1	2	1

Factorial

```
int fact(int n) {
  int r = 1;
  int i = 1;
  while (i <= n) {
    r = r * i;
    i = i + 1;
  }
  return r;
}
```

n	r	i	i <= n
5	1	1	1
5	1	2	1
5	2	3	1

Factorial

```
int fact(int n) {
  int r = 1;
  int i = 1;
  while (i <= n) {
    r = r * i;
    i = i + 1;
  }
  return r;
}
```

n	r	i	i <= n
5	1	1	1
5	1	2	1
5	2	3	1
5	6	4	1

Factorial

```
int fact(int n) {
  int r = 1;
  int i = 1;
  while (i <= n) {
    r = r * i;
    i = i + 1;
  }
  return r;
}
```

n	r	i	i <= n
5	1	1	1
5	1	2	1
5	2	3	1
5	6	4	1
5	24	5	1

Factorial

```
int fact(int n) {
  int r = 1;
  int i = 1;
  while (i <= n) {
    r = r * i;
    i = i + 1;
  }
  return r;
}
```

n	r	i	i <= n
5	1	1	1
5	1	2	1
5	2	3	1
5	6	4	1
5	24	5	1
5	120	6	0

Factorial

```
int fact(int n) {
  int r = 1;
  while (n > 0) {
    r = r * n;
    n = n - 1;
  }
  return r;
}
```

Factorial

```
int fact(int n) {
  int r = 1;
  while (n > 0) {
    r = r * n;
    n = n - 1;
  }
  return r;
}
```

Factorial

```
int fact(int n) {
  int r = 1;
  while (n > 0) {
    r = r * n;
    n = n - 1;
  }
  return r;
}
```

n	r	n > 0

Factorial

```
int fact(int n) {
  int r = 1;
  while (n > 0) {
    r = r * n;
    n = n - 1;
  }
  return r;
}
```

n	r	n > 0
5	1	1

Factorial

```
int fact(int n) {
  int r = 1;
  while (n > 0) {
    r = r * n;
    n = n - 1;
  }
  return r;
}
```

n	r	n > 0
5	1	1
4	5	1

Factorial

```
int fact(int n) {
  int r = 1;
  while (n > 0) {
    r = r * n;
    n = n - 1;
  }
  return r;
}
```

n	r	n > 0
5	1	1
4	5	1
3	20	1

Factorial

```
int fact(int n) {
  int r = 1;
  while (n > 0) {
    r = r * n;
    n = n - 1;
  }
  return r;
}
```

n	r	n > 0
5	1	1
4	5	1
3	20	1
2	60	1

Factorial

```
int fact(int n) {
  int r = 1;
  while (n > 0) {
    r = r * n;
    n = n - 1;
  }
  return r;
}
```

n	r	n > 0
5	1	1
4	5	1
3	20	1
2	60	1
1	120	1

Factorial

```
int fact(int n) {
  int r = 1;
  while (n > 0) {
    r = r * n;
    n = n - 1;
  }
  return r;
}
```

n	r	n > 0
5	1	1
4	5	1
3	20	1
2	60	1
1	120	1
0	120	0

Operadores de atribuição

```
var = var op expr;
```

≡

```
var op= expr;
```

Factorial

```
int fact(int n) {
  int r = 1;
  int i = 1;
  while (i <= n) {
    r *= i;
    i += 1;
  }
  return r;
}
```

Comandos vs Expressões

- Em C qualquer expressões pode ser usada como um comando

```
int main() {
  3+4;
  return 0;
}
```

- O comando de atribuição é de facto uma expressão
  - O valor da atribuição é o valor da expressão atribuída
  - O efeito no estado é modificar o valor da variável

Operadores ++ e --

Expressão	Valor	Efeito
`++x`	`x + 1`	`x = x + 1`
`x++`	`x`	`x = x + 1`
`--x`	`x - 1`	`x = x - 1`
`x--`	`x`	`x = x - 1`

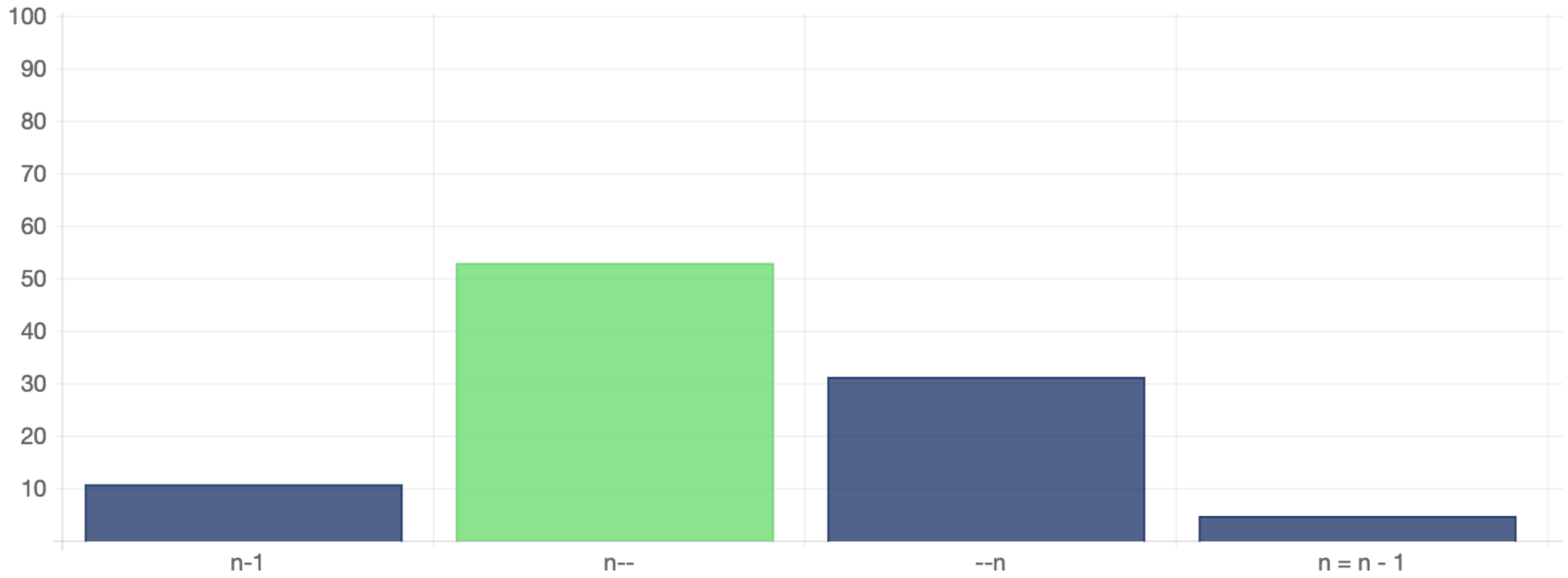
Factorial

```
int fact(int n) {
  int r, i;
  r = i = 1;
  while (i <= n) {
    r *= i;
    i++;
  }
  return r;
}
```

Que expressão usar para calcular o factorial?

```
int fact(int n) {
  int r = 1;
  while (n > 0) r *=      ;
  return r;
}
```

Que expressão usar para calcular o factorial?



Factorial

```
int fact(int n) {
  int r = 1;
  while (n -= (r *= n) > 0);
  return r;
}
```

Ciclo `for`

```
init; while (cond) {cmd; iter;}
```

≡

```
for (init; cond; iter) cmd;
```

Factorial

```
int fact(int n) {
  int r = 1;
  int i = 1;
  while (i <= n) {
    r *= i;
    i++;
  }
  return r;
}
```

```
int fact(int n) {
  int r = 1;
  for (int i = 1; i <= n; i++)
    r *= i;
  return r;
}
```

Factorial

```
int fact(int n) {
  int r = 1;
  while (n > 0) {
    r *= n;
    n--;
  }
  return r;
}
```

```
int fact(int n) {
  int r;
  for (r = 1; n > 0; n--)
    r *= n;
  return r;
}
```

Factorial

```
int fact(int n) {
  int r;
  for (r = 1; n > 0; r *= n, n--);
  return r;
}
```

Ciclo `do-while`

```
cmd; while (cond) cmd;
```

≡

```
do cmd; while (cond);
```

Factorial

```
int fact(int n) {
  int r = 1, i = 0;
  do {
    i++;
    r *= i;
  }
  while (i < n);
  return r;
}
```

Aula 4

`break` e `continue`

- O comando **`break`** termina a execução de um ciclo
  - A execução continua no primeiro comando depois do ciclo
- O comando **`continue`** termina a iteração actual
  - Num ciclo **`while`** ou **`do-while`** a execução continua no teste
  - Num ciclo **`for`** a execução continua no comando de incremento

Sorteio

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
  int r, i = 0;
  srand(time(NULL));        // inicializar o gerador de números pseudo-aleatórios
  while (i < 10) {
    r = rand();             // "sortear" um número entre 0 e RAND_MAX
    if (r == 0) break;
    if (r > 100) continue;
    printf("%d\n",r);
    i++;
  }
  return 0;
}
```

Sorteio

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
  int r, i = 0;
  srand(time(NULL));        // inicializar o gerador de números pseudo-aleatórios
  while (i < 10) {
    r = rand();             // "sortear" um número entre 0 e RAND_MAX
    if (r == 0) break;
    if (r <= 100) {
      printf("%d\n",r);
      i++;
    }
  }
  return 0;
}
```

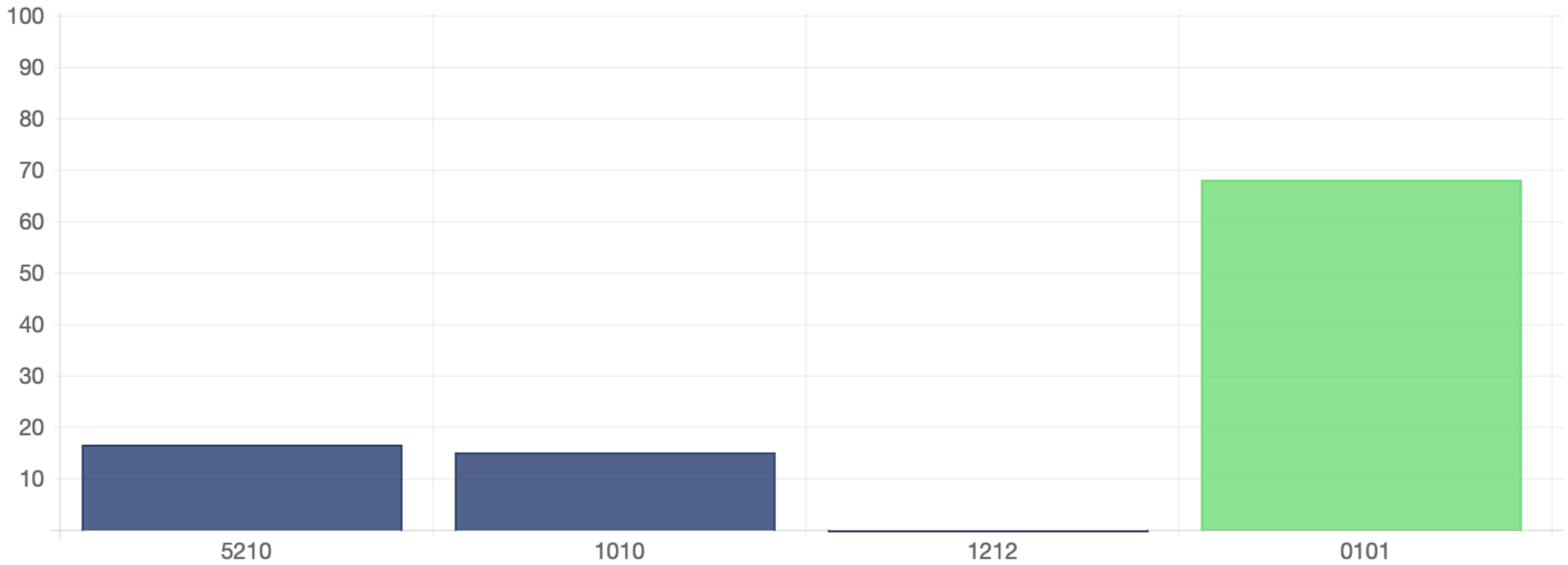
Sorteio

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
  int r, i = 0, ok = 1;
  srand(time(NULL));        // inicializar o gerador de números pseudo-aleatórios
  while (ok && i < 10) {
    r = rand();             // "sortear" um número entre 0 e RAND_MAX
    if (r == 0) ok = 0;
    if (ok && r <= 100) {
      printf("%d\n",r);
      i++;
    }
  }
  return 0;
}
```

Que imprime `proc(10)`?

```
void proc(unsigned int n) {
  for (; n > 0; n /= 2) printf("%d", n % 2);
}
```

Que imprime `proc(10)`?



Imprimir binário

```
void proc(unsigned int n) {
  int size = 0, m, i;
  for (m = n; m > 0; m /= 2) size++;
  for (size--; size >= 0; size--) {
    m = n;
    for (i = 0; i < size; i++) m /= 2;
    printf("%d", m % 2);
  }
}
```

Operadores lógicos *bitwise*

Operador	Significado	Exemplo (`unsigned char`)
`~`	Negação	`~10101001 == 01010110`
`&`	Conjunção	`10101001 & 11001010 == 10001000`
`|`	Disjunção	`10101001 | 11001010 == 11101011`
`^`	Disjunção exclusiva	`10101001 ^ 11001010 == 01100011`
`>>`	Shift para a direita	`10101001 >> 3 == 00010101`
`<<`	Shift para a esquerda	`10101001 << 3 == 01001000`

Imprimir binário

```
void proc(unsigned int n) {
  int size = 0, m;
  for (m = n; m > 0; m >>= 1) size++;
  for (size--; size >= 0; size--)
    printf("%d", (n >> size) & 1);
}
```

Recursividade

- Uma função pode invocar-se a si própria directa ou indirectamente

Factorial

```
int fact(int n) {
  int r;
  if (n > 0) r = n * fact(n-1);
  else r = 1;
  return r;
}
```

Imprimir binário

```
void proc(int n) {
  if (n > 0) {
    proc(n / 2);
    printf("%d", n % 2);
  }
}
```