# OPAKOVÁNÍ C

Stručně…

# Datové typy

- Datové typy určují velikost proměnné v paměti.

| Type | Storage size | Value range |
|---|---|---|
| char | 1 byte | -128 to 127 or 0 to 255 |
| unsigned char | 1 byte | 0 to 255 |
| signed char | 1 byte | -128 to 127 |
| int | 2 or 4 bytes | -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647 |
| unsigned int | 2 or 4 bytes | 0 to 65,535 or 0 to 4,294,967,295 |
| short | 2 bytes | -32,768 to 32,767 |
| unsigned short | 2 bytes | 0 to 65,535 |
| long | 8 bytes or (4bytes for 32 bit OS) | -9223372036854775808 to 9223372036854775807 |
| unsigned long | 8 bytes | 0 to 18446744073709551615 |

1 byte = 8 bitů
1 char = 1 byte = 8 bitů
1 char = např. 01001100 (L z ASCII tabulky)

void = datový typ bez velikosti
(ukazatel bez velikosti viz. dále)

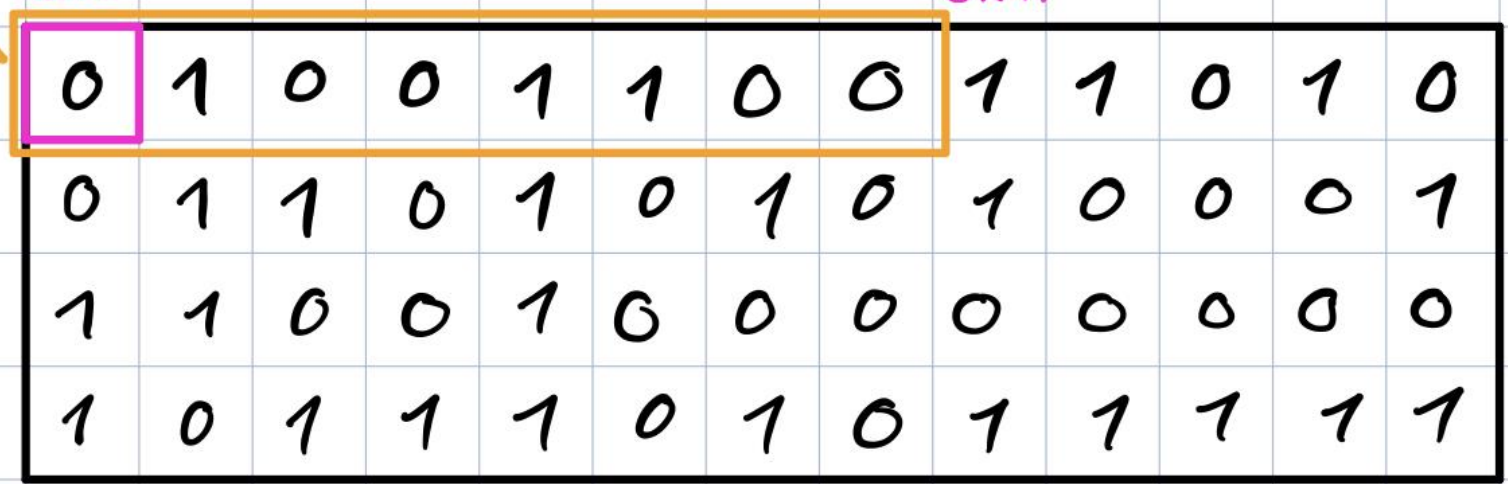
bin(11111111) = dec(255)
0-255 = 256 hodnot
$(11111111)_2 = (1 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = (255)_{10}$

C v základu nemá bool -> je potřeba importovat knihovnu stdbool.h

# Náčrtek paměti

```
char x = 'k';
```

- `char` – velikost: 1 char = 8 bit
- `x` – název
- `'k'` – hodnota: 01001100 → (k v ASCII)
- `&x` = adresa v paměti = 0x10




Paměť je rozdělena na dle velikosti nejmenšího datového typu (char = 1 byte)

0x700 → 0x701 mezi 700 a 701 je 1 byte = 8 bitů

# Proměnné a konstanty

- Proměnnou založíme takto:
  - typ název = hodnota;
    - int a = 10, b = 0;
    - float x, y;
    - char c = ‘T‘;

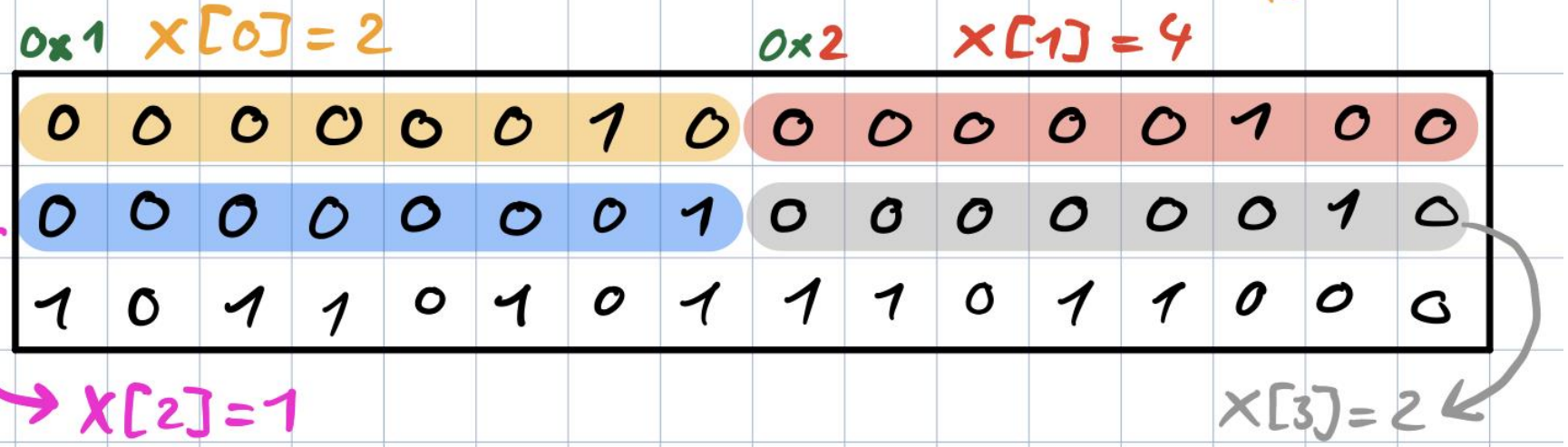
&a = adresa prvního bitu proměnné a (v paměti)

- Konstantu založíme takto:
  - const typ název = hodnota; (konstantní proměnná, dá se změnit přes ukazatel)
    - const int X = 100;
    - const char C = ‘K‘;
  - #define název hodnota
    (řeší preprocesor, neukládá se do paměti)
    - #define X 100
    - #define C ‘K‘

pole je řada čísel → 2, 4, 1, 2

`char x[4] = {2,4,1,2};` = pole čtyř bytů

1 char = 1 byte




pokud chceme třeba druhý prvek z pole tak máme dvě možnosti

1) `x[1]`
2) `*(x+1)`

x je ukazatel na první prvek v poli
↳ x = 0x1

↳ hodnotu ukazatele získáme pomocí * → *x = 2

# Základní struktura

```c
#include <stdio.h>

int main() {
    /* my first program in C */
    printf("Hello, World! \n");

    return 0;
}
```

Přidání základní knihovny
(musí být vždy pokud chceme používat nějaké funkce)

Hlavní funkce main
(musí se vždy jmenovat main)

Samotný program, který se spustí.
- výpis textu „Hello, World!“
- \n je znak pro nový řádek (enter)

Návratová hodnota funkce main
(0 = vše ok)

# Funkce

```c
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

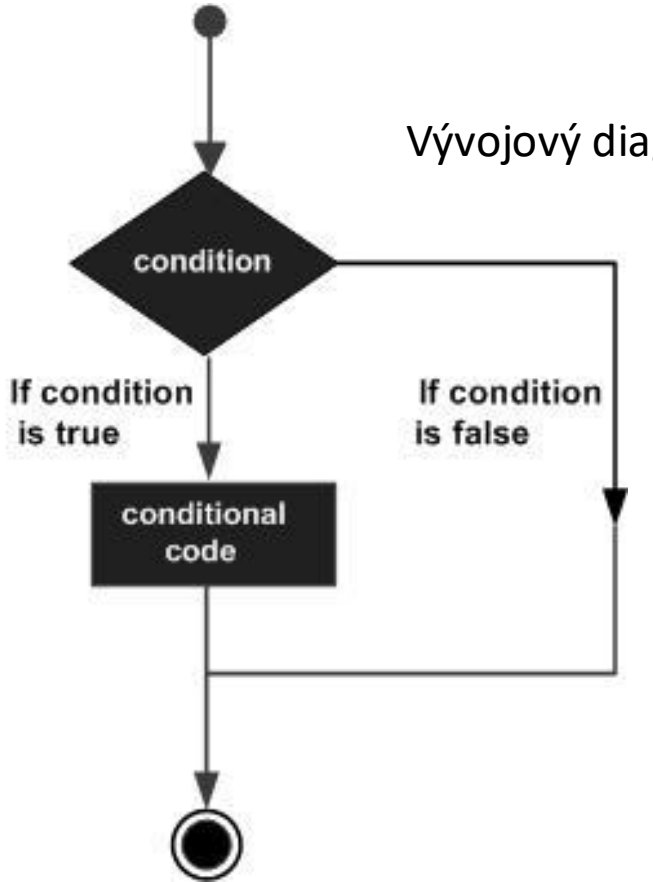
- Deklarace funkce max, která vrací int a požaduje dva int při volání (num1 a num2).
  - Deklarací řekneme, že taková funkce existuje.
  - Místo deklarace můžeme hned funkci definovat (tím spojíme definici a deklaraci dohromady)(méně přehledné).
  - Pokud chceme zavolat funkci max, tak musí byt vždy **deklarována** nad (před) místem, kde jí chceme zavolat (v tomto případě před voláním max ve funkci main musela proběhnout deklarace funkce max).
  - Datový typ „int“ před názvem funkce nám říká jaký bude datový typ hodnoty, kterou vrátí.
    - Můžeme tedy mít i char max(), long max(), float max(), atd…
    - Funkce s datovým typem void nevrací žádnou hodnotu.

- Definice funkce max.
  - Definicí specifikujeme co funkce dělá.
  - Proměnné, které založíme ve funkci jsou viditelné jen ve funkci, kde byly založeny.
  - Vstupní proměnné se chovají jako normální proměnné definované uvnitř funkce (jejich hodnota se nastaví při volání funkce).

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 100;

   /* check the boolean condition */
   if( a == 10 ) {
      /* if condition is true then print the following */
      printf("Value of a is 10\n" );
   } else if( a == 20 ) {
      /* if else if condition is true */
      printf("Value of a is 20\n" );
   } else if( a == 30 ) {
      /* if else if condition is true  */
      printf("Value of a is 30\n" );
   } else {
      /* if none of the conditions is true */
      printf("None of the values is matching\n" );
   }

   printf("Exact value of a is: %d\n", a );

   return 0;
}
```

# Podmínky (if, else if, else)

- Podmínky využíváme k realizaci logických funkcí.
- Jak bude vypadat vývojový diagram pro if-else a if-else if-else?

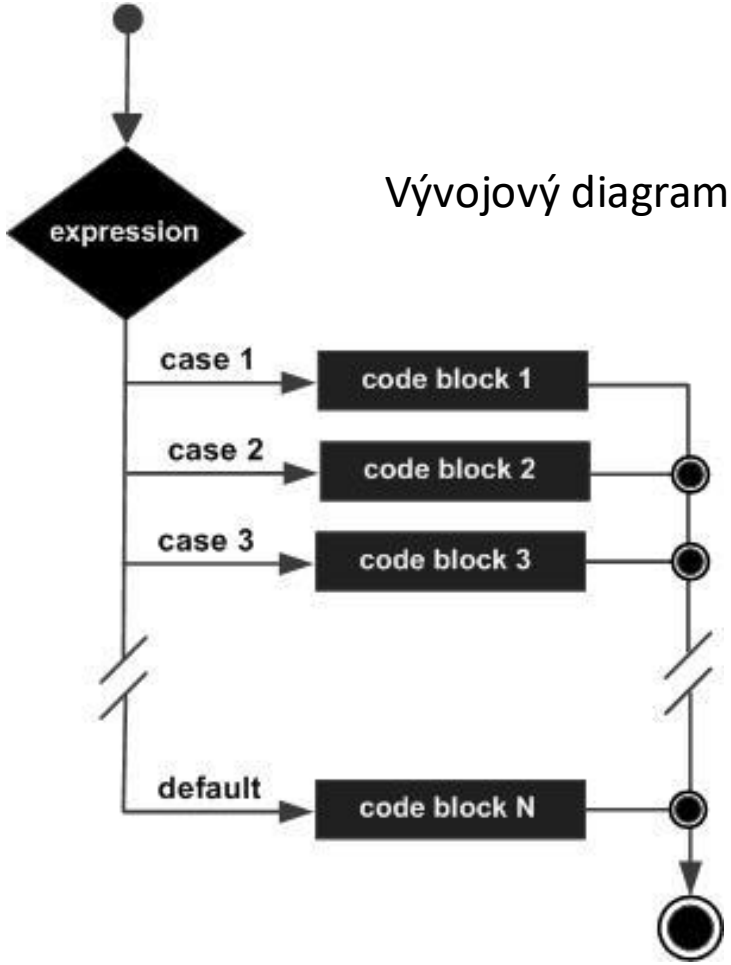



Vývojový diagram pro if.

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   char grade = 'B';

   switch(grade) {
      case 'A' :
         printf("Excellent!\n" );
         break;
      case 'B' :
      case 'C' :
         printf("Well done\n" );
         break;
      case 'D' :
         printf("You passed\n" );
         break;
      case 'F' :
         printf("Better try again\n" );
         break;
      default :
         printf("Invalid grade\n" );
   }

   printf("Your grade is  %c\n", grade );

   return 0;
}
```

# Podmínky (switch)

- Switch použijeme pokud potřebujeme porovnat proměnnou s více hodnotami.
- POZOR! Funguje pouze jako základní porovnání (==). Nelze tedy použít operátory <, >, <=, >=, atd…
- Switch se dá vždy přepsat na if, ale obráceně to už neplatí. Proč?



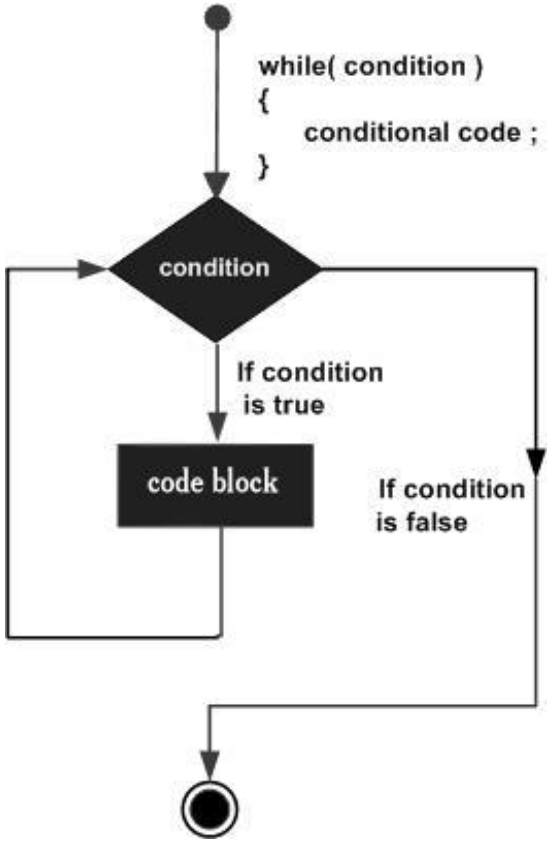

Vývojový diagram pro switch.

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
}
```

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {
      printf("value of a: %d\n", a);
      a = a + 1;
   }while( a < 20 );

   return 0;
}
```

# Cykly (while, do while)

- Cykly while a do while budou provádět svojí část kódu dokud jejich podmínka bude pravdivá.
- Jak vytvoříme nekonečný cyklus?
  - Nekonečný program (cyklus) můžete ukončit stisknutím Ctrl+C.




Cyklus lze ovlivnit následujícími příkazy:

- break – zastaví cyklus a bude pokračovat v kódu za cyklem
- continue – přeskočí aktuální iteraci cyklu
  - Lze tak například přeskočit výpočet 1/x když se x = 0, aby program nevyhodil chybu.

Vývojový diagram pro while.
Jak bude vypadat vývojový diagram pro do while?

# Cykly (for)

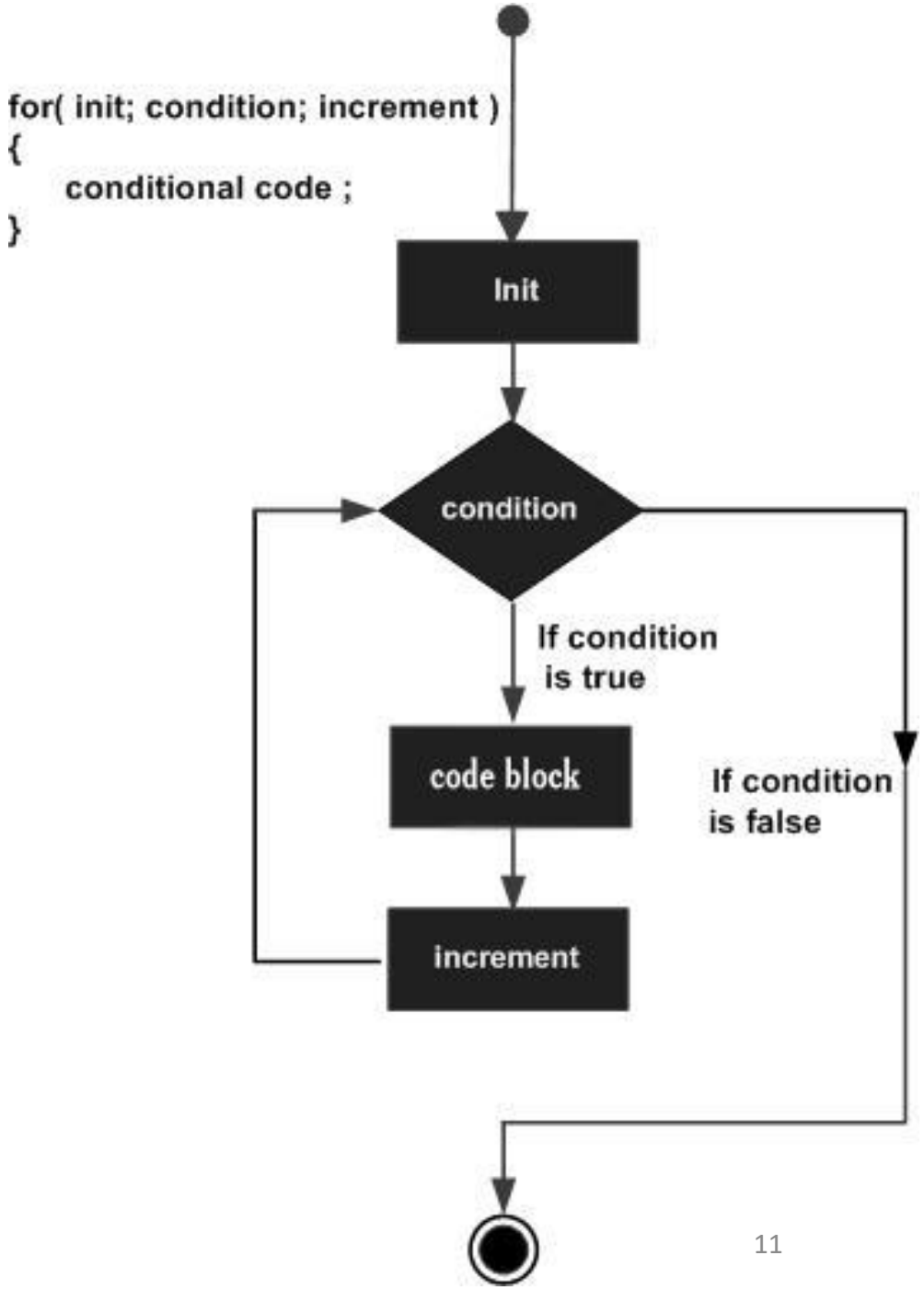
- Cyklus for se hodí na procházení polí.
- Každý for se dá přepsat na while a obráceně.
- Lze také využívat break a continue.

Program:

```c
#include <stdio.h>

int main () {

   int a;

   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 ){
      printf("value of a: %d\n", a);
   }

   return 0;
}
```

Výstup:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```



# Odkazy

- Tutorial na C (výrazně doporučeno)
- Specifikace funkce printf (formátovací znaky)
- Online GDB