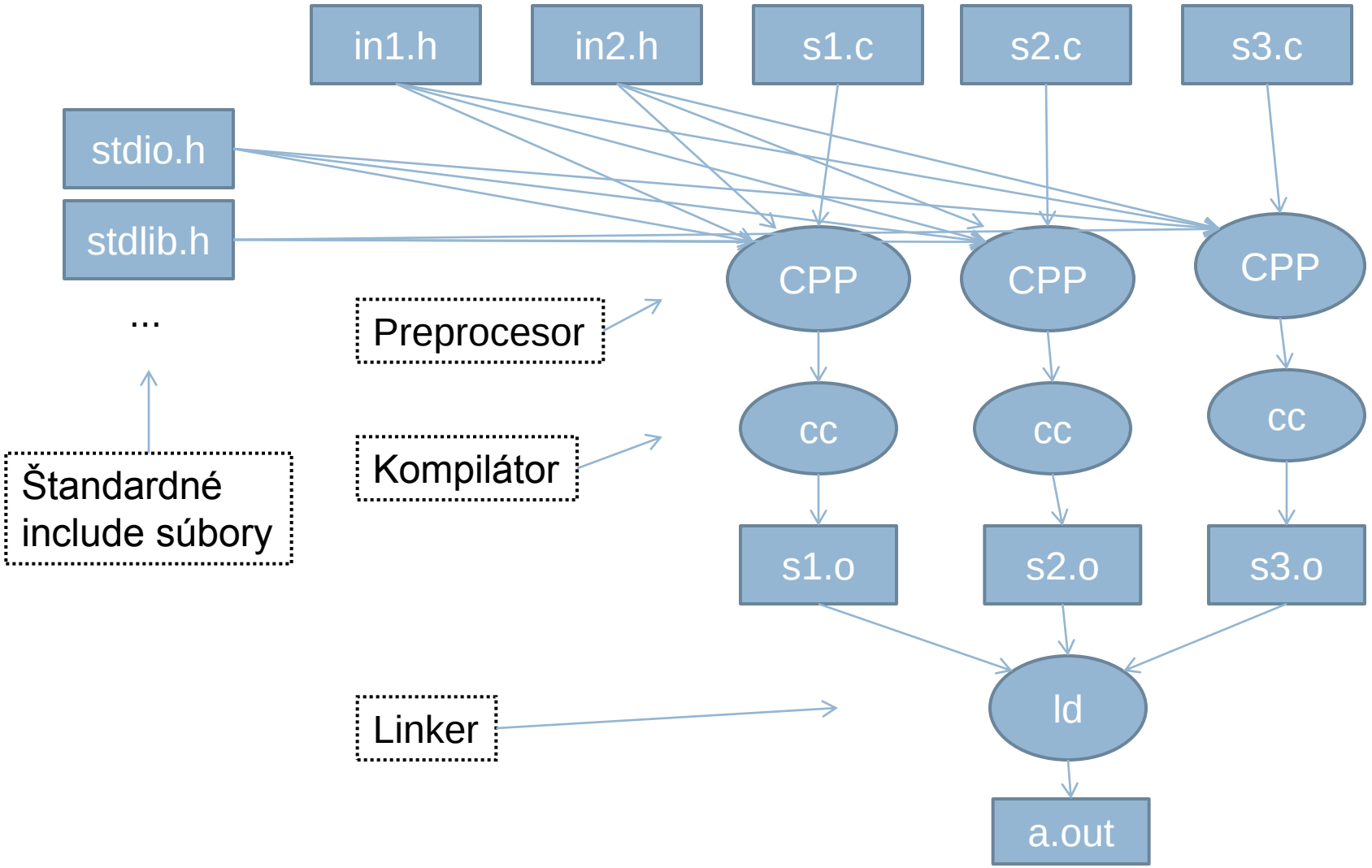
VYBRANÉ PARTIE Z JAZYKA C

C

- Program == dáta + algoritmus

- Program je kolekcia funkcií / procedúr.
- Dátové typy sú priamo inšpirované hardwarom počítačov.
- Stručná syntax, preprocessor, ktorý je aj nie je súčasťou jazyka.
- Je to kompilovaný jazyk. Program sa kompiluje optimalizujúcim kompilátorom do binárneho kódu priamo vykonávaným daným hardware-om.

Architektúra kompilácie



Funkcie a procedúry

- Parametre, návratová hodnota
- Telo funkcie pozostáva z definície lokálnych premenných a postupnosti príkazov. Klasické matematické výrazy, priradenie, vetvenie (if, switch), cykly (for, while, do), príkaz skoku goto.

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

int faktorial(int n) {
  int i, r;
  r = 1;
  for(i=2; i<=n; i++) {
    r = r*i;
  }
  return(r);
}

void main() {
  printf("%d\n", faktorial(9));
}
```

Len pre ilustráciu: text z ANSI štandardu popisujúci základné typy:

There are five standard signed integer types, designated as signed char, short int, int, long int, and long long int. (These and other types may be designated in several additional ways, as described in 6.7.2.) There may also be implementation-defined extended signed integer types. The standard and extended signed integer types are collectively called signed integer types.

An object declared as type signed char occupies the same amount of storage as a "plain" char object. A "plain" int object has the natural size suggested by the architecture of the execution environment (large enough to contain any value in the range INT_MIN to INT_MAX as defined in the header <limits.h>.

For each of the signed integer types, there is a corresponding (but different) unsigned integer type (designated with the keyword unsigned) that uses the same amount of storage (including sign information) and has the same alignment requirements. The type _Bool and the unsigned integer types that correspond to the standard signed integer types are the standard unsigned integer types. The unsigned integer types that correspond to the extended signed integer types are the extended unsigned integer types. The standard and extended unsigned integer types are collectively called unsigned integer types.

Aritmetické dátové typy

- Základné aritmetické typy: očakávaná veľkosť v bytoch
  - char, alebo signed char 1
  - unsigned char 1
  - short, alebo short int, signed short, signed short int 2
  - unsigned short, alebo unsigned short int 2
  - int, alebo signed int 4
  - unsigned, alebo unsigned int 4
  - long, alebo long int, signed long int 4
  - unsigned long, alebo unsigned long int 4
  - long long int, alebo signed long long int 4
  - unsigned long long, alebo unsigned long long int 8
  - float 4
  - double 8
  - long double 16

Bežné súčasné implementácie

	LP32	ILP32	ILP64	LLP64	LP64
Char	8	8	8	8	8
Short	16	16	16	16	16
Int	16	32	64	32	32
Long	32	32	64	32	64
Long long	64	64	64	64	64
pointer	32	32	64	64	64

Ako zistiť na akej implementácii pracujete?

- Operátor **`sizeof`** vráti veľkosť (v bytoch) pre akýkoľvek typ alebo výraz.

- Má dva tvary:
  - **`sizeof`**`(`*`typ`*`)`
  - **`sizeof`** *`výraz`*

Ak potrebujete použiť typy s presnou veľkosťou

- Súbor `stdint.h` definuje typy `intN_t` a `uintN_t` pre N = 8, 16, 32, 64, ?.

```
#include <stdio.h>
#include <stdint.h>

int main() {
    uint32_t    x;
    int8_t      a;
    printf("%d %d\n", sizeof(x), sizeof(a));
}
```

Vypíše: **4 1**

```c
#include <stdio.h>

int main() {
      uint8_t   a;
      char      b;

      a == 130;
      b = a;
      printf("a == %d; b == %d\n", a, b);
}
```

```c
#include <stdio.h>

int main() {
	char		a;

	a = 0x81;
	if (a == 0x81) printf("Ano\n");
	else printf("Nie\n");
}
```

Reálne číselné typy

float	4
double	8
long double	8 /16 (10)

Všeobecne sa predpokladá, že aritmetické operácie nad reálnymi číslami sú výrazne pomalšie ako nad celočíselnými typmi.

Niektoré reálne čísla sa nedajú v počítači presne reprezentovať, treba si pri nich dávať pozor.

```
float f = 0.7;
if (f == 0.7) printf("Ano");
else printf("Nie");
```

```
double f = 0.7;
if (f == 0.7) printf("Ano");
else printf("Nie");
```

```
double f = 0.7;
if (2 * f == 1.4) printf("Ano");
else printf("Nie");
```

```
double f = 0.7;
if (3 * f == 2.1) printf("Ano");
else printf("Nie");
```

Odvodené dátové typy

- Smerníky.
- Polia.
- Štruktúry.
- Uniony.

Smerníky (pointre)

- Pointer je vlastne celé číslo, ktoré sa používa na adresáciu v pamäti. Je 32-bitový na 32 bitových systémoch a 64-bitový na 64 bitových systémoch.
- V C sa smerník (pointer) viaže k nejakému typu (budeme ho nazývať *základný* typ) z ktorého je odvodený ("na ktorý ukazuje"). Presnejšie povedané, obsah pamäte na danej adrese sa interpretuje ako premenná (objekt) daného *základného* typu.

Pointre

- Definícia premennej cez operátor *

```
int *a;
```

- Prístup k hodnote. Ak p je pointer na typ t, tak *p je výraz typu t.
- Smerníková aritmetika. Pripočítanie jednotky ku smerníku zväčší adresu o veľkosť (sizeof) základného typu na ktorý smerník ukazuje. Toto pravidlo je konzistentne rozšírené na pripočítanie (odpočítanie) akejkoľvek celočíselnej hodnoty.

Postfixový operátor [ ]

- postfixový operátor [ ] (ktorý slúži na indexovanie) má nasledovnú definíciu: Ak **e1** a **e2** sú výrazy, tak **e1[e2]** je ekvivalentné výrazu ***(e1 + (e2))**.

```
#include <stdio.h>
int main() {
    char    *p = "ahoj";
    int     i;
    for(i=0; p[i]!=0; i++) {
      printf("%d\n",p[i]);
    }
}
```

p08.c

Polia

- súvislá oblasť pamäte vyplnená prvkami s daným typom

```
int a[10];        // pole desiatich prvkov typu int
double b[20];     // pole 20 realnych čísel
int x[10][20];    // pole 10 polí o 20 prvkoch int
int y[2][3][4]; // pole 2 polí 3 polí 4 prvkov ...
```

Polia

- Pole sa dá initializovať cez {}

```
int a[100] = {0,1,2,3,4};
int b[] = {2,3,4};
```

- Pole znakov sa dá initializovať cez ""

```
char s[100] = "Hello";
char t[] = "Bonjour";
```

- Pole sa nedá poslať (hodnotou) ako parameter funkcie! Dá sa uviesť formálny parameter typu pole, ale znamená to, že parametrom je pointer na prvý prvok poľa.

Polia vo výrazoch

- sizeof pre pole vráti počet bytov ktoré zaberá celé pole.
- Unárny operátor & aplikovaný na pole vráti pointer na prvý prvok poľa.
- vo všetkých iných výrazoch sa pole správa (je konvertované) ako konštantný pointer na prvý prvok poľa. Presne ako keby ste naň vždy aplikovali operátor &.
  - Napríklad pri použití operátora [ ]

```
int i, a[100];
for(i=0; i<100; i++) a[i] = 0;
```

```c
#include <stdio.h>

int main() {
      int a[5];
      int b[] = {2,3};
      int *c;

      char *p = "Nazdar sudruhovia";
      char q[] = "Zdar sudruh general";

      printf("%d\n", sizeof(a));
      printf("%d\n", sizeof(b));
      printf("%d\n", sizeof(c));
      printf("%d\n", sizeof(p));
      printf("%d\n", sizeof(q));
}
```

Zradný príklad

```
#include <stdio.h>

void fun(char p[10]) {
     printf("%p %p\n", p, &p);
}

int main() {
     char a[10];
     printf("%p %p\n", a, &a);
     fun(a);
}
```

Nezradný príklad ekvivalentný s predchádzajúcim zradným

```
#include <stdio.h>

void fun(char *p) {
     printf("%p %p\n", p, &p);
}

int main() {
     char a[10];
     printf("%p %p\n", a, &a);
     fun(a);
}
```

Ešte zradnejší príklad

```
#include <stdio.h>

void fun(char p[10]) {
     printf("%d\n", sizeof(p));
}

int main() {
     char a[10];
     printf("%d\n", sizeof(a));
     fun(a);
}
```

Viacrozmerné polia (polia polí)

```
int A[3][5];
```

A == A[0]	A [0] [0]	A [0] [1]	A [0] [2]	A [0] [3]	A [0] [4]
A[1]	A [1] [0]	A [1] [1]	A [1] [2]	A [1] [3]	A [0] [4]
A[2]	A [2] [0]	A [2] [1]	A [2] [2]	A [2] [3]	A [2] [4]

```c
#include <stdio.h>

int main() {
  char a[3][5];
  printf("%d %d\n", sizeof(a), sizeof(a[0]));
}
```

Pointer na pole

```
#include <stdio.h>

void fun(char (*b)[5]) {
     printf("%d %d %d\n", b, b+1, sizeof(b));
}

int main() {
     char a[3][5];
     printf("%d %d %d\n", a, a+1, sizeof(a));
     fun(p);
}
```

Pointer na pole 2

```
#include <stdio.h>

void fun(char b[3][5]) {
     printf("%d %d\n", b, sizeof(b));
}

int main() {
     char a[3][5];
     printf("%d %d\n", a, sizeof(a));
     fun(p);
}
```

Pointer na pole 3

```
#include <stdio.h>

void fun(char b[][5]) {
     printf("%d %d\n", b, sizeof(b));
}

int main() {
     char a[3][5];
     printf("%d %d\n", a, sizeof(a));
     fun(p);
}
```

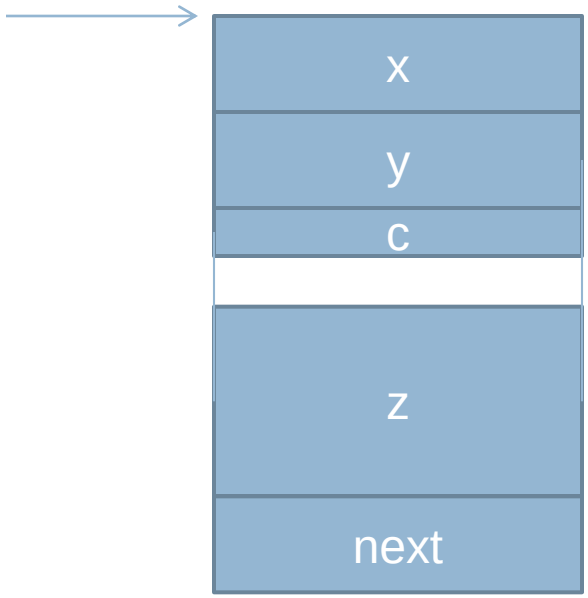
Viacrozmerné pole je stále len pole

```
void fun(char **b) {
}

void gun(char b[][]) { // chyba
}

int main() {
   char a[3][5];
   fun(a);                    // chyba
}
```

Štruktúry

- postupnosť objektov s možno odlišnými typmi.

```
struct sss {
    int             x,y;
    char            c;
    double          z;
    struct sss      *next;
};

struct sss *p;
```



Bitové polia

- Bitové polia sa definujú ako špecialny prípad štruktúry, kde celočíselné typy majú veľkosť s presne daným počtom bitov.

```
struct cas {
  unsigned minuta:6;
  unsigned hodina:5;
  unsigned den:5;
  unsigned mesiac:4;
  unsigned rok:12;
};
```

Uniony

- Syntax pred definovanie unionu je rovnaká ako pre štruktúru, ale všetky členy sú uložené na tej istej adrese na začiatku unionu. Slúži to na ušetrenie zmeny typov (castov). Veľkosť unionu je určená veľkosťou najväčšieho členu.

```
struct vrcholStromu {
    int     somList;
    union {
        double                  hodnota;
        struct VrcholStromu     podstromy[2];
    } u;
};
```

Deklarácie, definície

Preprocessor