



PROGRAMOVACIE JAZYKY PRE VSTAVANÉ SYSTÉMY

Úvod

ÚVODNÉ INFORMÁCIE

- Zoznámenie
- Motivácia
- Obsah predmetu
- Požiadavky na cvičenia a skúšku



PREDSTAVENIE

- Prednášajúci/cvičiaci:
- Miroslav Kvaššay
- Katedra informatiky
- Miroslav.Kvassay@fri.uniza.sk



MOTIVÁCIA

- Prečo tento predmet?



LITERATÚRA

- B. Kernighan, D. Ritchie: Programovací jazyk C, ALFA+SNTL, 1998
- P. Herout: Učebnice jazyka C (1. díl), Kopp, 2009
- P. Martincová, K. Grondžák, M. Zábovský: Programovanie v jadre operačného systému Linux, EDIS, 2008
- Zdroje na internete
 - <http://en.cppreference.com/w/c/language>
 - <http://www.cplusplus.com/reference/clibrary/>



VSTAVANÝ SYSTÉM

- Počítačový systém, vstavaný do väčšieho elektrického, resp. mechanického systému.
- Často má charakter systému s reálnym časom.
- Požiadavky:
 - Nízka spotreba
 - Malé rozmery
 - Nízka cena



VSTAVANÝ SYSTÉM

○ Hardvér:

- Mikrokontroléry
- Bežné CPU

○ Použitie:

- Inteligentné hodinky, MP3 prehrávače
- Riadenie križovatiek, dopravné značky
- Hybridné automobily, aviatika



VSTAVANÝ SYSTÉM

- História:
- 60. roky minulého storočia – kozmický program
- Integrované obvody
- Mikrokontroléry
- Súčasnosť - RISC procesory, ARM,...



VSTAVANÝ SYSTÉM

- Používateľské rozhranie:
 - Žiadne viditeľné rozhranie
 - Jednoduché indikátory (LED, tlačidlá,...)
 - Riadkové displeje
 - Grafické displeje, dotykové obrazovky

- Komunikácia:
 - RS232
 - USB
 - I2C
 - Ethernet



VSTAVANÝ SYSTÉM

- Architektúry vstavaných systémov:
 - **Jednoduchá slučka** – zariadenie kontroluje vstupy a podľa stavu ich obsluhuje
 - **Prerušením riadený systém**
 - **Kooperatívny multitasking** – starší spôsob práce OS
 - **Preemptívny multitasking** – jadro prepína medzi úlohami



VSTAVANÝ SYSTÉM

- Vývoj aplikácií pre vstavané systémy:
 - Najčastejšie sa využíva programovací jazyk C
 - Existujú špecifiká vývoja aplikácií pre vstavané systémy:
 - Limitované možnosti procesora,
 - Prenositelnosť
 - Komunikácia s používateľom



JAZYK C - ÚVOD

- Je to procedurálny programovací jazyk
- Používaný pre nízkoúrovňové programovanie
- Používa konštrukcie, ktoré sa ľahko mapujú na inštrukcie procesora
- Populárny pre tvorbu operačných systémov, ovládačov zariadení, programovanie firmvéru pre vstavané systémy,...



HISTÓRIA

- Vytvorený v rokoch 1969 – 1973 v AT&T Bell Labs, pôvodne ako nástroj pre napísanie jadra operačného systému Unix a ako jeho programovací nástroj.
- Hlavný tvorca – **Dennis Ritchie** (1941 – 2011) (vpravo). Vľavo **Ken Thompson** (1943) – jeden z tvorcov OS Unix.

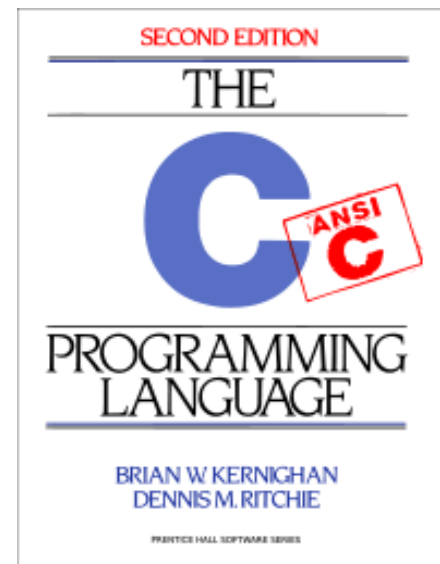
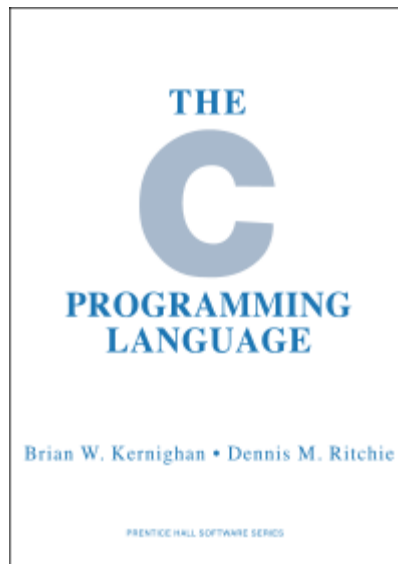


(Zdroj: Wikipedia)



ŠTANDARDIZÁCIA

- 1978 – prvá špecifikácia: Brian Kernighan and Dennis Ritchie: The C Programming Language, **K&R C**.
- 1989: ANSI X3.159-1989 Programming Language C, **ANSI C, C89**
- 1990: ISO/IEC 9899:1990, **C90**
- 1999: ISO/IEC 9899:1999 **C99**
- 2011: ISO/IEC 9899:2011, **C11**



(Zdroj: Wikipedia)



SYNTAX

- Jazyk C je definovaný formálnou gramatikou
- Projekt sa skladá z deklarácií a definícií premenných a z definícií funkcií
- Funkcia obsahuje definície premenných a príkazy
- Je definovaná špeciálna funkcia, od ktorej sa začína vykonávanie programu – funkcia **main**
- Používaná množina znakov:
 - malé a veľké písmená
 - číslice
 - špeciálne znaky: ! " # % & ' () * + , - . / : ; < = > ? [\] ^ _ { | } ~
 - biele znaky: medzera, horizontálny tabulátor, vertikálny tabulátor, posun stránky (FF), návrat vozíka (CR)



KLÚČOVÉ SLOVÁ

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
inline (C99)	int	long	register
restrict (C99)	return	short	signed
sizeof	static	struct	switch
typedef	union	unsigned	void
volatile	while	_Alignas (C11)	_Alignof (C11)
_Atomic (C11)	_Bool (C99)	_Complex (C99)	_Generic (C11)
_Imaginary (C99)	_Noreturn (C11)	_Static_assert (C11)	_Thread_local (C11)

vrátane všetkých názvov začínajúcich znakom `_`, za ktorým nasleduje veľké písmeno alebo znak `_`



UKÁŽKA ZDROJOVÉHO KÓDU V JAZYKU C

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  /*
6   * Funkcia vypocita obsah trojuholnika na zaklade Heronovho vzorca.
7   */
8  double obsahTrojuholnika(double a, double b, double c) {
9      double s = (a + b + c) / 2.0;
10     return sqrt(s*(s-a)*(s-b)*(s-c));
11 }
12
13 int main(int argc, char** argv) {
14     double a, b, c; //premenne predstavujuce strany trojuholnika
15
16     printf("Zadaj stranu a:\n");
17     scanf("%lf", &a);
18     printf("Zadaj stranu b:\n");
19     scanf("%lf", &b);
20     printf("Zadaj stranu c:\n");
21     scanf("%lf", &c);
22     printf("%f", a);
23     printf("Obsah trojuholnika je %.4f a jeho obvod je %.4f.\n",
24           obsahTrojuholnika(a, b, c), a + b + c);
25     return 0;
26 }
```



ZDROJOVÝ KÓD

- Zdrojový kód programu v jazyku C pozostáva z:

- **deklarácií** (ukončené ;):

```
void faktorial(int n);
```

- **príkazov** (ukončené ; alebo ohraničené { }):

```
x = 5 + y;  
{  
    x = 6 * x;  
}
```

- **komentárov:**

- viacriadkový:

```
/*Toto je  
viacriadkový  
komentár*/
```

- jednoriadkový (**C99**):

```
//Toto je jednoriadkový komentár
```

- **direktív preprocesora** (začínajú znakom #):

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



IDENTIFIKÁTOR

- Označuje entitu v programe (premenná, údajový typ, funkcia, návestie, makro, parameter makra).
- Je tvorený postupnosťou písmen, číslíc a znaku `_`, pričom nesmie začínať číslicom (a nemal by začínať ani znakom `_`).
 - Mnohé prekladače umožňujú použiť v identifikátore aj symbol `$` (možný warning).
- Dĺžka nie je obmedzená (teoreticky).
- Identifikátory sú case sensitive.
- Kľúčové slová nemôžu byť použité ako identifikátor.
- Napr.: `sirka`, `obvodStvorca`, `x_123`.



DEKLARÁCIA A DEFINÍCIA

- **Deklarácia** – konštrukt zavádzajúci do programu jeden alebo viacero nových identifikátorov, pričom špecifikuje ich význam (premenná, funkcia, nový dátový typ) a vlastnosti, napr.:
 - **extern** int a; (~~extern int a = 100;~~)
 - int sucet(int, int);
 - int sucet(int a, int b);
 - struct bod;
- **Definícia** – deklarácia, ktorá obsahuje všetky informácie potrebné pre vytvorenie deklarovaného identifikátora (premenná – v ktorej časti pamäti bude uložená, funkcia – ako pracuje, dátový typ – akú má veľkosť a aký je význam jednotlivých bitov), napr.:
 - int a;
 - int a = 100;
 - int sucet(int a, int b) { return a + b; }
 - struct bod { double x, double y};



OBLASŤ PLATNOSTI IDENTIFIKÁTOROV

- Identifikátory môžu byť viditeľné na úrovni:
 - **bloku**
 - lokálne premenné
 - formálne parametre funkcie
 - dátové typy
 - **súboru**
 - globálne premenné
 - funkcie
 - dátové typy
 - **funkcie**
 - návestia
 - **funkčného prototypu**
 - formálne parametre funkcie



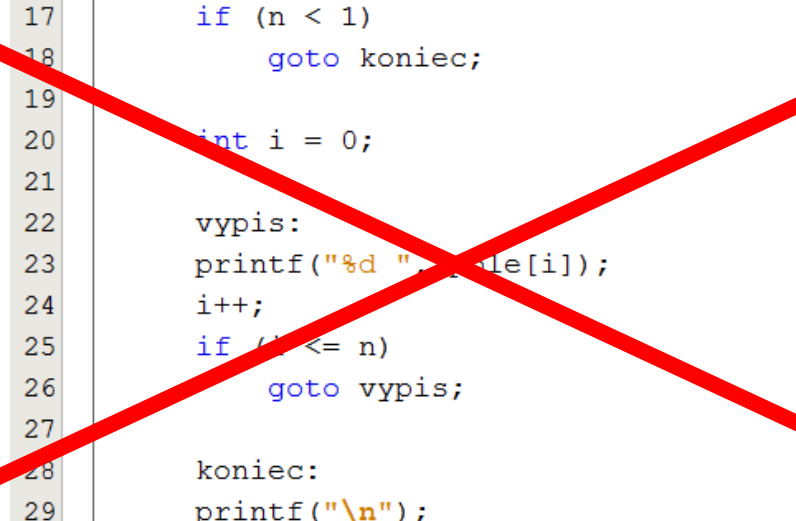
OBLASŤ PLATNOSTI IDENTIFIKÁTOROV – PRÍKLAD

```
5  const int velkost = 10;
6
7  void vypisCisla(int n, int pole[n]);
8
9  □ int main(int argc, char** argv) {
10     int pole[velkost];
11
12     vypisCisla(velkost, pole);
13     return 0;
14 }
15
16 □ void vypisCisla(int n, int pole[n]) {
17     if (n < 1)
18         goto koniec;
19
20     int i = 0;
21
22     vypis:
23     printf("%d ", pole[i]);
24     i++;
25     if (i <= n)
26         goto vypis;
27
28     koniec:
29     printf("\n");
30 }
```



OBLASŤ PLATNOSTI IDENTIFIKÁTOROV – PRÍKLAD

```
5  const int velkost = 10;
6
7  void vypisCisla(int n, int pole[n]);
8
9  int main(int argc, char** argv) {
10     int pole[velkost];
11
12     vypisCisla(velkost, pole);
13     return 0;
14 }
15
16 void vypisCisla(int n, int pole[n]) {
17     if (n < 1)
18         goto koniec;
19
20     int i = 0;
21
22     vypis:
23     printf("%d ", pole[i]);
24     i++;
25     if (i <= n)
26         goto vypis;
27
28     koniec:
29     printf("\n");
30 }
```



ŠTANDARDNÉ DÁTOVÉ TYPY (1)

○ Základné (aritmetické):

- znak (**char**)
- celočíselné znamienkové (signed char, short, **int**, long, **long long** (C99))
- celočíselné neznamienkové (_Bool (C99), unsigned char, unsigned short, unsigned int, unsigned long, unsigned long long (C99))
- s pohyblivou desatinnou čiarkou:
 - reálne: **float**, **double**, long double
 - komplexné (C99) (float _Complex, double _Complex, long double _Complex)
 - imaginárne (C99) (float _Imaginary, double _Imaginary, long double _Imaginary)



ŠTANDARDNÉ DÁTOVÉ TYPY (2)

- Vymenovaný typ (**enum**)
- Typ **void**
- Odvodené:
 - **pole**
 - štruktúra (**structure**)
 - zjednotenie (**union**)
 - **ukazovateľ**
 - atomické typy (C11) (**_Atomic**) – aplikovateľné na ľubovoľný dátový typ okrem poľa



PREMENNÉ

- Premenná asociuje miesto v pamäti s identifikátorom.
- Pomocou premenných sa odvolávame na príslušné miesto v pamäti, kde je uložená hodnota daného typu.
- Môže existovať viacero **deklarácií** premennej (kľúčové slovo **extern**):
 - `extern <typ> <zoznam deklarátorov>;`
 - `extern int a, b;`
- Premenná môže byť **definovaná** maximálne jeden raz:
 - `<typ> <zoznam deklarátorov>;`
 - `int a;`
 - `int b = 10, c = 20;`
 - `extern int d = 30;` (len v prípade globálnych premenných; nepoužívať – zhoršuje čitateľnosť kódu)



KONŠTANTY A LITERÁLY

- Predstavujú konštantné hodnoty zapísané v programe.
- Konštanty (r-hodnoty):
 - celočíselné
 - reálne
 - znakové
- Literály (l-hodnoty):
 - reťazcové
 - zložené („compound“) (C99) – anonymné polia, štruktúry a union-y



CELOČÍSELNÉ KONŠTANTY

- Desiatková sústava: 127, 5563987
- Osmičková sústava: **012**, **0123456**
- Šestnástková sústava: **0xFF**, **0xff**, **0XDEADBEEF**
- Dátový typ konštanty je **najbližší celočíselný typ** (minimálne však int), do ktorého sa hodnota zmestí, ak nemá príponu:
 - u, U – unsigned: 123u
 - l, L – long: 123L
 - ll, LL – long long: 123LL
 - kombinácie – unsigned long, unsigned long long: 123uL



REÁLNE KONŠTANTY

- Desiatková sústava: 6.023e23, 5.02E18, 1., .123, 1.602e-19 ($e = E = 10^{\wedge}$)
- Šestnástková sústava (C99): 0x1.FFp14, 0x1.2P3, 0x1.2p-14 ($p = P = 2^{\wedge}$)
- Konštanta je typu **double**, ak nemá príponu:
 - f, F – float
 - l, L – long double
- Určte, akého typu budú nasledujúce konštanty:
 - 48.2L
 - 48.L
 - 48L



ZNAKOVÉ KONŠTANTY

- Jeden znak v apostrofoch: 'a', 'U', '1',...
- Konštanta je typu **int** (v C++ char), ak nemá predponu:
 - u (C11) – char16_t: u'\uf34c'
 - U (C11) – char32_t: U'\U0001f34c'
 - L – wchar_t: L'ß'
- Escape sequence:
 - '\\', '\"', '\?', '\\\\', '\\a', '\\b', '\\f', '\\n', '\\r', '\\t', '\\v'
 - '\\nnn' – osmičkový zápis ASCII hodnoty znaku: '\\0'
 - '\\xnn' – šestnástkový zápis ASCII hodnoty znaku: '\\x41'
 - '\\u_{nnnn}', '\\U_{nnnnnnnn}' (C99) – šestnástkový zápis Unicode hodnoty znaku



REŤAZCOVÉ LITERÁLY

- Postupnosť znakov uzavretých v úvodzovkách:
 - "ahoj"
- Spájanie:
 - "ahoj" " Peter\n"
- Reťazcový literál je implementovaný ako pole znakov, ktorého posledný prvok je znak '\0'.
- Jednotlivé prvky literálu sú modifikovateľné l-hodnoty, avšak ich modifikácia povedie k chybe v programe (napr. „segmentation fault“):

~~"ahoj"[0] = 'X'~~



FUNKCIE

- Funkcia – konštrukt jazyka C, ktorý asociuje zložený príkaz (telo funkcie) s identifikátorom (názov funkcie).
- Funkcia môže mať 0 alebo viac parametrov. Ak sa neuvedie typ parametru, tak je typu int.
- Funkcia môže vracať hodnotu. Ak sa neuvedie návratový typ, funkcia vracia int. Ak funkcia nemá vracať hodnotu musíme použiť návratový typ void.
- Jazyk C nepodporuje vnorené funkcie.
- Príkaz return bez parametrov spôsobí ukončenie vykonávania funkcie a vrátenie riadenia do volajúceho kódu.
- Počet príkazov return vo funkcii nie je obmedzený.



DEKLARÁCIA A DEFINÍCIA FUNKCIE (OD C89)

- Definícia:

```
double obsahTrojuholnika(double a, double b, double c) {  
    double s = (a + b + c) / 2.0;  
    return sqrt(s*(s-a)*(s-b)*(s-c));  
}
```

- Deklarácia:

```
double obsahTrojuholnika(double a, double b, double c);
```

```
double obsahTrojuholnika(double, double, double);
```



DEKLARÁCIA A DEFINÍCIA FUNKCIE (K&R)

○ Definícia:

```
double obsahTrojuholnika(a, b, c)
double a; double b; double c;
{
    double s = (a + b + c) / 2.0;
    return sqrt(s*(s-a)*(s-b)*(s-c));
}
```

○ Deklarácia:

```
double obsahTrojuholnika();
```

- Ak chceme deklarovať funkciu, ktorá nemá žiadne parametre, tak píšeme:

```
double obsahTrojuholnika(void);
```



FUNKCIA MAIN

- Funkcia main predstavuje vstupný bod programu.
- Ak funkcia neobsahuje príkaz return, vráti sa hodnota 0.
- Syntax:

```
int main(void) { telo funkcie }
```

```
int main(int argc, char *argv[]) { telo funkcie }
```

```
int main(int argc, char *argv[], dalsie_parametre) { telo funkcie }
```



PRÍKAZ

- Pokyn na vykonanie nejakej činnosti.

- Jazyk C pozná 5 typov príkazov:

- (výrazový) **príkaz**:

`n = faktorial(5) + x;`

- **zložený príkaz (blok)**:

```
{  
    x = 3 + y;  
    y++;  
}
```

- **príkazy vetvenia** (if-else, switch)
- **príkazy cyklov** (for, while, do-while)
- **príkazy skoku** (break, continue, return, goto)
- Hociktorý príkaz môže byť označený 1 alebo viacerými návestiami (popiskami, „label“):

`fakt: n = faktorial(5) + x;`



VÝRAZY (1)

- Postupnosť operátorov a ich operandov, ktorá definuje výpočet, napr.:
 - $x + 3$
 - $\text{pow}(2, x) + 7 * y$
 - $\text{printf}(\text{"\%d\\n"}, \text{sqrt}(z + 4))$
- Výrazy môžu byť **objektového** alebo **funkčného** typu.
- Každý výraz patrí do jednej z nasledujúcich kategórií:
 - l-hodnota – výraz objektového typu, ktorý zodpovedá identite objektu; na l-hodnotu je možné aplikovať operátor &
 - modifikovateľná – výraz môže stáť na ľavej strane operátora priradenia
 - r-hodnota – výraz objektového typu, ktorý neoznačuje objekt, ale skôr predstavuje konkrétnu hodnotu, ktorú môže mať nejaký objekt
 - „function designator“ – výraz funkčného typu



VÝRAZY (2)

- Predpokladajme nasledujúce premenné:
 - `int i;`
 - `float x;`
 - `int pole[100];`
- Určte, ktoré z nasledujúcich výrazov sú l-hodnoty:
 - `i`
 - `(i + 7) * 3`
 - `i + x`
 - `(x)`
 - `pole[20]`
 - `pole[2 * i + 1]`
 - `pole`
 - `24`
 - `"ahoj"`



OPERÁTORY

- Výrazy konštruujeme pomocou operátorov.
- Operátory môžeme rozdeliť do niekoľkých skupín:
 - podľa **počtu operandov** – unárne, binárne, ternárne
 - podľa **pozície vo výraze** – prefixové, infixové, postfixové
 - podľa **typu operácie**:
 - aritmetické (+ (unárne/binárne), - (unárne/binárne), *, /, %)
 - bitové (~, <<, >>, &, |, ^)
 - inkrementačné/dekrementačné (++ (prefixové/postfixové), -- (prefixové/postfixové))
 - priradovacie (=, +=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^=)
 - logické (!, &&, ||)
 - relačné (==, !=, >, <, >=, <=)
 - prístupové ([] , ->, ., *, &)
 - iné (), (type), sizeof, _Alignof (C11),?:, ,)



PRIORITA OPERÁTOROV

Operátor	Asociativita
postfixové ++ postfixové -- () [] . ->	Zľava doprava
prefixové ++ prefixové -- unárne + unárne - ! ~ (type) * & sizeof _Alignof	Sprava doľava
* / %	Zľava doprava
binárne + binárne -	Zľava doprava
<< >>	Zľava doprava
< <= > >=	Zľava doprava
== !=	Zľava doprava
&	Zľava doprava
^	Zľava doprava
	Zľava doprava
&&	Zľava doprava
	Zľava doprava
?:	Sprava doľava
= += -= *= /= %= <<= >>= &= ^= =	Sprava doľava
,	Zľava doprava



OPERÁTOR PRIRADENIA

- Pre výraz $A = B$ platí:
 - ak výrazy A a B sú rôzneho typu tak B je implicitne skonvertované na typ A (ak je to možné)
 - výraz A musí byť modifikovateľná l-hodnota
 - výraz $A = B$ vracia hodnotu, ktorá bola uložená do A , preto môžeme písať:
 - $A = B = C$
- Obdobné tvrdenia platia aj pre zložené operátory priradenia.
- Určte, ktoré z nasledujúcich výrazov sú korektné:
 - $A = (B = (C = 2))$
 - $A = (B = C) = 2$
 - $A *= B *= C *= 2$



RELAČNÉ A LOGICKÉ OPERÁTORY

- Relačné operátory (`==`, `!=`, `>`, `>=`, `<`, `<=`):
 - výraz obsahujúci relačný operátor má typ **int** a jeho hodnota je **1**, ak príslušná relácia platí, alebo **0**, ak príslušná relácia neplatí
 - ľavý a pravý operand operátorov `==` a `!=` musí byť číselný typ (vrátane `_Complex` a `_Imaginary`) alebo smerník
 - ľavý a pravý operand operátorov `>`, `>=`, `<` a `<=` musí byť reálny číselný typ (t.j. nie `_Complex` alebo `_Imaginary`) alebo smerník
- Logické operátory (`!`, `&&`, `||`):
 - výraz obsahujúci logický operátor má typ **int** a jeho hodnota je **1**, ak príslušná podmienka platí, alebo **0**, ak neplatí
 - operandmi môžu byť číselné typy (vrátane `_Complex` a `_Imaginary`) alebo smerník
 - operátory `&&` a `||` majú skrátené vyhodnocovanie
 - vo výrazoch **A && B** a **A || B** sa po výraze A nachádza „sequence point“
- Určte, akú hodnotu má nasledujúci výraz:
 - `!(10 > 1 || !(10.5) != 1)`



ARITMETICKÉ VÝRAZY – URČENIE TYPU

- Výsledný typ aritmetického výrazu závisí od typu jednotlivých operandov:
 - ak majú operandy rovnaký typ, typ výrazu sa zhoduje s typom operandov.
 - ak je typ operandov rôzny, dôjde k implicitným konverziám, po ktorých budú mať operandy rovnaký typ (výnimku predstavujú výrazy s operandmi typ `_Complex` alebo `_Imaginary`).
- Majme nasledujúce premenné:
 - `signed char c = 10;`
 - `short s = -5;`
 - `int i = 5;`
 - `unsigned int j = 7;`
 - `float f = 3.1415926;`
- Určte, akého typu budú nasledujúce výrazy:
 - `i + 10`
 - `f / j + 12 * i`
 - `i / j + s`
 - `s * c`
 - `s / i + i / (f * s)`



ARITMETICKÉ VÝRAZY – PRAVIDLÁ PRE URČENIE TYPU („USUAL ARITHMETIC CONVERSIONS“) (1)

1. Ak jeden z operandov je **long double**, **long double _Complex** alebo **long double _Imaginary**, druhý je skonvertovaný nasledovne:
 - celočíselné typy a reálne typy na **long double**
 - komplexný typ na **long double _Complex**
 - imaginárny typ na **long double _Imaginary**
2. Inak, ak jeden z operandov je **double**, **double _Complex** alebo **double _Imaginary**, druhý je skonvertovaný nasledovne:
 - celočíselné typy a typ **float** na **double**
 - komplexný typ na **double _Complex**
 - imaginárny typ na **double _Imaginary**
3. Inak, ak jeden z operandov je **float**, **float _Complex** alebo **float _Imaginary**, druhý je skonvertovaný nasledovne:
 - celočíselné typy na **float**
 - komplexný typ zostáva **float _Complex**
 - imaginárny typ zostáva **float _Imaginary**
4. Inak, oba operandy sú celočíselné.



ARITMETICKÉ VÝRAZY – PRAVIDLÁ PRE URČENIE TYPU („USUAL ARITHMETIC CONVERSIONS“) (2)

5. Ak oba operandy sú celočíselné, tak
1. operandy s rozsahom menším alebo rovným typu **int** (**_Bool**, **char**, **short**, **unsigned short**,...) sú skonvertované na typ **int** alebo **unsigned int**
 2. typ operandov sa skonvertuje podľa nasledujúcej tabuľky

	ull	ll	ul	l	u	int
ull	ull	ull	ull	ull	ull	ull
ll		ll	ll (Windows 64) ull (Unix 64)	ll	ll (Windows 64, Unix 64) ull	ll
ul			ul	ul	ul	ul
l				l	l (Unix 64) ul (Windows 64)	l
u					u	u
int						int



ARITMETICKÉ VÝRAZY – PRAVIDLÁ PRE URČENIE TYPU („USUAL ARITHMETIC CONVERSIONS“) (3)

- Určte typy nasledujúcich výrazov:
 - $1.f + 123456789$
 - $10L + (\text{char})'b'$
 - $10u - 50$
 - $0UL - 1LL$

 - `double _Complex z = 1 + 2*I;`
 - $z + 1.0$

- Vyššie popísané implicitné konverzie sa aplikujú na operandy aj v prípade **relačných operátorov**, **bitových operácií** a **ternárneho operátora**.
 - Určte typ nasledujúceho výrazu:
 - $a < b ? 1.1 : 10$



DIREKTÍVA #INCLUDE

- Slúži na vloženie iného súboru do aktuálneho zdrojového súboru. Obsah vkladaneého súboru sa vloží hneď za direktívu.
- Syntax:
 #include <subor>
 #include "subor"



ŠTANDARDNÁ KNIŽNICA JAZYKA C

- Samotný jazyk neposkytuje nástroje pre vstup/výstup, matematické funkcie, prácu s reťazcami, ...
- Existuje štandardná C knižnica, štandardizovaná ISO a ANSI:
 - `stdio.h`
 - `stdlib.h`
 - `math.h`
 - ...
- Pre UNIX-like systémy sú k dispozícii volania jadra, POSIX:
 - `pthread.h`
 - `arpa/inet.h`
 - `sys/socket.h`
 - ...



FUNKCIE PRINTF A SCANF

- `printf(formatovaci_retazec[, argumenty])` – formátovaný výstup na konzolu

```
int a, b;
```

```
double c, d;
```

```
printf("Sucet %d a %d je %d\n", a, b, a + b);
```

```
printf("Sucet %.2f a %.2f je %.2f\n", c, d, c + d);
```

- `scanf(formatovaci_retazec[, argumenty])` – formátovaný vstup z klávesnice

```
int a, b;
```

```
double c, d;
```

```
scanf("%d%d", &a, &b);
```

```
scanf("%lf%lf", &c, &d);
```

- Hlavičkový soubor: `stdio.h`



ORGANIZÁCIA PROJEKTU (1)

- Projekt v jazyku C/C++:
- Zdrojové kódy - *.c
- Hlavičkové súbory – *.h, prototypy funkcií, globálne konštanty,...
- Knižnice
- Z týchto častí vytvoríme výslednú spustiteľnú aplikáciu



ORGANIZÁCIA PROJEKTU (2)

