7. lekce

Úvod do jazyka C – 1. část

– knihovny
– datové typy, definice proměnných

Miroslav Jílek

https://en.cppreference.com – internetová stránka s referencemi

https://gedit.en.softonic.com/download – download editoru zdrojového kódu „Gedit“

http://tdm-gcc.tdragon.net/download - dawnload kompilátoru gcc

Zrození jazyka C

„Céčko“ navrhl Dennis Ritchie, tvůrce operačního systému UNIX,

Bellovy laboratoře, Murray Hill, New Jersey (USA), rok 1972

Výhody jazyka C:

Výhody jazyka C:

- extrémně rychlý

Výhody jazyka C:

- extrémně rychlý
- multiplatformní (pracuje na většině operačních systémů)

Výhody jazyka C:

- extrémně rychlý
- multiplatformní (pracuje na většině operačních systémů)

Nevýhody jazyka C:

Výhody jazyka C:

- extrémně rychlý
- multiplatformní (pracuje na většině operačních systémů)

Nevýhody jazyka C:

- nízký komfort pro programátora

Výhody jazyka C:

- extrémně rychlý
- multiplatformní (pracuje na většině operačních systémů)

Nevýhody jazyka C:

- nízký komfort pro programátora
- neumí pracovat s textovými řetězci

Výhody jazyka C:

- extrémně rychlý
- multiplatformní (pracuje na většině operačních systémů)

Nevýhody jazyka C:

- nízký komfort pro programátora
- neumí pracovat s textovými řetězci
- nemá grafické rozhraní

Výhody jazyka C:

- extrémně rychlý
- multiplatformní (pracuje na většině operačních systémů)

Nevýhody jazyka C:

- nízký komfort pro programátora
- neumí pracovat s textovými řetězci
- nemá grafické rozhraní
- nepodporuje objektově orientované programování

Výhody jazyka C:

- extrémně rychlý
- multiplatformní (pracuje na většině operačních systémů)

Nevýhody jazyka C:

- nízký komfort pro programátora
- neumí pracovat s textovými řetězci
- nemá grafické rozhraní
- nepodporuje objektově orientované programování

Speciální vlastnost:

- přímý přístup do paměti (rychlejší přístup, ale nutno pečlivě pracovat s deklarací proměnných)

Procesy od programování do spuštění programu:

Procesy od programování do spuštění programu:

1) vytvoření zdrojového kódu – v programovacím jazyce C

Procesy od programování do spuštění programu:

1) vytvoření zdrojového kódu – v programovacím jazyce C

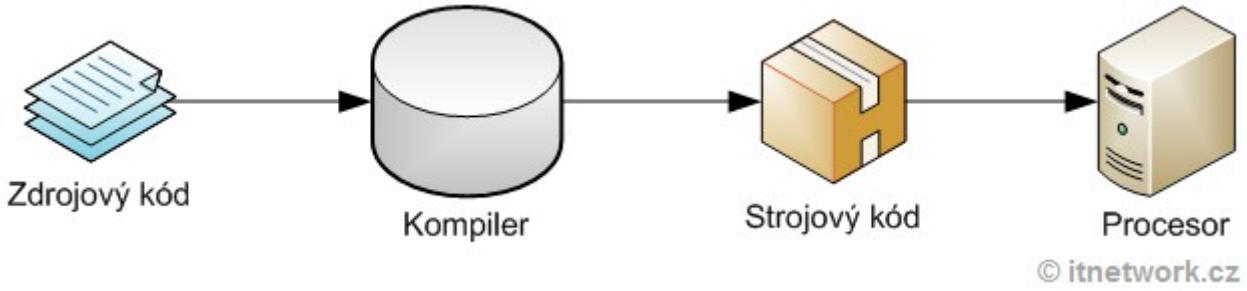
2) zkompilování do strojového kódu (pouze v případě bezchybného kódu)

Procesy od programování do spuštění programu:

1) vytvoření zdrojového kódu – v programovacím jazyce C

2) zkompilování do strojového kódu (pouze v případě bezchybného kódu)

3) Spuštění strojového kódu v procesoru (na operačním systému)



Připojení systémových knihoven

Systémové knihovny umožňují využívat různé funkce jazyka C...

Připojení systémových knihoven

Systémové knihovny umožňují využívat různé funkce jazyka C...

Zápis do kódu programu:

```
#include <knihovna>
```

Připojení systémových knihoven

Systémové knihovny umožňují využívat různé funkce jazyka C...

Zápis do kódu programu:

#include <knihovna>

knihovny: stdio.h - vstupy a výstupy

Připojení systémových knihoven

Systémové knihovny umožňují využívat různé funkce jazyka C...

Zápis do kódu programu:

#include <knihovna>

knihovny: stdio.h - vstupy a výstupy
stdlib.h - standardní knihovna se základními instrukcemi

Připojení systémových knihoven

Systémové knihovny umožňují využívat různé funkce jazyka C...

Zápis do kódu programu:

#include <knihovna>

knihovny: stdio.h - vstupy a výstupy
stdlib.h - standardní knihovna se základními instrukcemi
math.h - matematické funkce

Připojení systémových knihoven

Systémové knihovny umožňují využívat různé funkce jazyka C...

Zápis do kódu programu:

#include <knihovna>

knihovny:
stdio.h - vstupy a výstupy
stdlib.h - standardní knihovna se základními instrukcemi
math.h - matematické funkce
assert.h - funkce pro kontrolu programu

Připojení systémových knihoven

Systémové knihovny umožňují využívat různé funkce jazyka C...

Zápis do kódu programu:

#include <knihovna>

knihovny:

stdio.h	- vstupy a výstupy
stdlib.h	- standardní knihovna se základními instrukcemi
math.h	- matematické funkce
assert.h	- funkce pro kontrolu programu
string.h	- funkce pro práci s polem znaků (char) jako se stringem

Připojení systémových knihoven

Systémové knihovny umožňují využívat různé funkce jazyka C...

Zápis do kódu programu:

#include <knihovna>

knihovny:
- stdio.h - vstupy a výstupy
- stdlib.h - standardní knihovna se základními instrukcemi
- math.h - matematické funkce
- assert.h - funkce pro kontrolu programu
- string.h - funkce pro práci s polem znaků (char) jako se stringem

Do kódu programu knihovny zapisujeme na začátek - před hlavní program.

Základní datové typy

Základní datové typy

Každá proměnná musí být deklarovaná v proceduře dříve, než je napsán první příkaz!

Základní datové typy

Každá proměnná musí být deklarovaná v proceduře dříve, než je napsán první příkaz!

Pozor: Při definici proměnné je důležité, jestli použijeme malá nebo velká písmena v názvech proměnných nebo funkcí! Proměnná **a** není proměnná **A**!

Základní datové typy

Každá proměnná musí být deklarovaná v proceduře dříve, než je napsán první příkaz!

Pozor: Při definici proměnné je důležité, jestli použijeme malá nebo velká písmena v názvech proměnných nebo funkcí! Proměnná **a** není proměnná **A**!

int integer (celé číslo –2^32 až +2^32 – 1)
int A; nebo ***int A = hodnota;***

Pokud není uvedeno rovnítko a hodnota, pak proměnná má nedefinovanou hodnotu, tedy nějakou hodnotu, kterou náhodně obsahuje paměť, kterou operační systém přidělil programu a program přidělil proměnné. To může způsobit problémy při chodu programu a je tedy nutné dát pozor na přidělení hodnot proměnným dříve, než je budeme zpracovávat v programu!

Základní datové typy

Každá proměnná musí být deklarovaná v proceduře dříve, než je napsán první příkaz!

Pozor: Při definici proměnné je důležité, jestli použijeme malá nebo velká písmena v názvech proměnných nebo funkcí! Proměnná **a** není proměnná **A**!

int integer (celé číslo –2^32 až +2^32 – 1)
int A; nebo ***int A = hodnota;***

Pokud není uvedeno rovnítko a hodnota, pak proměnná má nedefinovanou hodnotu, tedy nějakou hodnotu, kterou náhodně obsahuje paměť, kterou operační systém přidělil programu a program přidělil proměnné. To může způsobit problémy při chodu programu a je tedy nutné dát pozor na přidělení hodnot proměnným dříve, než je budeme zpracovávat v programu!

char jeden znak (reprezentován číslem od -128 do +127)

Základní datové typy

Každá proměnná musí být deklarovaná v proceduře dříve, než je napsán první příkaz!

Pozor: Při definici proměnné je důležité, jestli použijeme malá nebo velká písmena v názvech proměnných nebo funkcí! Proměnná **a** není proměnná **A**!

int integer (celé číslo –2^32 až +2^32 – 1)
int A; nebo ***int A = hodnota;***

Pokud není uvedeno rovnítko a hodnota, pak proměnná má nedefinovanou hodnotu, tedy nějakou hodnotu, kterou náhodně obsahuje paměť, kterou operační systém přidělil programu a program přidělil proměnné. To může způsobit problémy při chodu programu a je tedy nutné dát pozor na přidělení hodnot proměnným dříve, než je budeme zpracovávat v programu!

char jeden znak (reprezentován číslem od -128 do +127)

double desetinné číslo (pozor: desetinná tečka!)

Základní datové typy

Každá proměnná musí být deklarovaná v proceduře dříve, než je napsán první příkaz!

Pozor: Při definici proměnné je důležité, jestli použijeme malá nebo velká písmena v názvech proměnných nebo funkcí! Proměnná **a** není proměnná **A**!

int integer (celé číslo –2^32 až +2^32 – 1)
int A; nebo ***int A = hodnota;***

Pokud není uvedeno rovnítko a hodnota, pak proměnná má nedefinovanou hodnotu, tedy nějakou hodnotu, kterou náhodně obsahuje paměť, kterou operační systém přidělil programu a program přidělil proměnné. To může způsobit problémy při chodu programu a je tedy nutné dát pozor na přidělení hodnot proměnným dříve, než je budeme zpracovávat v programu!

char jeden znak (reprezentován číslem od -128 do +127)

double desetinné číslo (pozor: desetinná tečka!)

float desetinné číslo s menším rozsahem (pozor: desetinná tečka!)

boolean používá se proměnná typu integer; 0 je false, jiné číslo je true!!!

boolean používá se proměnná typu integer; 0 je false, jiné číslo je true!!!

string neexistuje, použije se pole char

boolean používá se proměnná typu integer; 0 je false, jiné číslo je true!!!

string neexistuje, použije se pole char

pole definice: datový_typ jméno_proměnné[počet prvků]
např. ***int pole[10];*** – pole deseti prvků, první index je nula, poslední devět!

Proměnná typu pointer

- pointer = ukazatel, ukazuje na místo v RAM

Proměnná typu pointer

- pointer = ukazatel, ukazuje na místo v RAM
- obsahuje adresu do RAM, která ukazuje na hodnotu, která je uložena na daném místě v RAM

Proměnná typu pointer

- pointer = ukazatel, ukazuje na místo v RAM
- obsahuje adresu do RAM, která ukazuje na hodnotu, která je uložena na daném místě v RAM
- hodnota proměnné typu pointer je adresa v RAM

Proměnná typu pointer

- pointer = ukazatel, ukazuje na místo v RAM
- obsahuje adresu do RAM, která ukazuje na hodnotu, která je uložena na daném místě v RAM
- hodnota proměnné typu pointer je adresa v RAM
  např. 0A75ff12 (4 byty u 32 bitového systému)
  0A75ff12224ccc87 (8 bytů u 64 bitového systému)
  1 znak může obsahovat symboly 0 až f a to je číslo 0 až 15, v binárním kódu 0000 až 1111 – jeden znak potřebuje 4 bity, proto u 8 znaků potřebujeme 32 bitů a to jsou 4 byty

Proměnná typu pointer

- pointer = ukazatel, ukazuje na místo v RAM
- obsahuje adresu do RAM, která ukazuje na hodnotu, která je uložena na daném místě v RAM
- hodnota proměnné typu pointer je adresa v RAM
  např. 0A75ff12 (4 byty u 32 bitového systému)
  0A75ff12224ccc87 (8 bytů u 64 bitového systému)
  1 znak může obsahovat symboly 0 až f a to je číslo 0 až 15, v binárním kódu 0000 až 1111 – jeden znak potřebuje 4 bity, proto u 8 znaků potřebujeme 32 bitů a to jsou 4 byty

Vlastnosti pointerů

- šetří operační paměť (při kopírování proměnné můžeme použít jenom její adresu v RAM)

Proměnná typu pointer

- pointer = ukazatel, ukazuje na místo v RAM
- obsahuje adresu do RAM, která ukazuje na hodnotu, která je uložena na daném místě v RAM
- hodnota proměnné typu pointer je adresa v RAM
  např. 0A75ff12 (4 byty u 32 bitového systému)
  0A75ff12224ccc87 (8 bytů u 64 bitového systému)
  1 znak může obsahovat symboly 0 až f a to je číslo 0 až 15, v binárním kódu 0000 až 1111 – jeden znak potřebuje 4 bity, proto u 8 znaků potřebujeme 32 bitů a to jsou 4 byty

Vlastnosti pointerů

- šetří operační paměť (při kopírování proměnné můžeme použít jenom její adresu v RAM)
- umožňují přístup ke stejnému místu v paměti přes různé proměnné

Definice proměnné typu pointer

datový_typ *jmeno_proměnné

Definice proměnné typu pointer

datový_typ *jmeno_proměnné

Např.:

int *promenna1; proměnná, které obsahuje adresu v paměti RAM, kde je uložena hodnota typu integer.
Pokud chceme přistoupit k hodnotě, která je uložena pod daným pointerem, pak před jménem proměnné zapíšeme hvězdičku!
*Hodnota proměnné **promenna1** je adresa do paměti RAM, na které je v paměti uložena hodnota (např. nějaké číslo).*

Definice proměnné typu pointer

datový_typ *jmeno_proměnné

Např.:

int *promenna1; proměnná, které obsahuje adresu v paměti RAM, kde je uložena hodnota typu integer.
Pokud chceme přistoupit k hodnotě, která je uložena pod daným pointerem, pak před jménem proměnné zapíšeme hvězdičku!
*Hodnota proměnné **promenna1** je adresa do paměti RAM, na které je v paměti uložena hodnota (např. nějaké číslo).*

Použití pointeru:

promenna1 = 10;

Znamená, že jsme na adresu uloženou v proměnné ***promenna1*** zapsali hodnotu 10.

Definice proměnné typu pointer

datový_typ *jmeno_proměnné

Např.:

int *promenna1; proměnná, které obsahuje adresu v paměti RAM, kde je uložena hodnota typu integer.
Pokud chceme přistoupit k hodnotě, která je uložena pod daným pointerem, pak před jménem proměnné zapíšeme hvězdičku!
*Hodnota proměnné **promenna1** je adresa do paměti RAM, na které je v paměti uložena hodnota (např. nějaké číslo).*

Použití pointeru:

promenna1 = 10;

Znamená, že jsme na adresu uloženou v proměnné ***promenna1*** zapsali hodnotu 10.

Pokud chceme získat adresu proměnné a uložit ji do pointeru, pak před jméno proměnné zapíšeme & (bez mezery):
promenna1 = &A;

NULL je speciální hodnota ukazatele. Znamená, že ukazatel ukazuje nikam – neznáme adresu místa v RAM!

NULL je speciální hodnota ukazatele. Znamená, že ukazatel ukazuje nikam – neznáme adresu místa v RAM!

Např.: *int *x=NULL;*
**x=10;*
Chyba – nelze vložit hodnotu na neexistující adresu! *(nevíme, kam vložit....)*

NULL je speciální hodnota ukazatele. Znamená, že ukazatel ukazuje nikam – neznáme adresu místa v RAM!

Např.: *int *x=NULL;*
**x=10;*
Chyba – nelze vložit hodnotu na neexistující adresu! *(nevíme, kam vložit....)*

Příklad použití pointeru:

```
int a, *p;
a = 56;
printf("Promenna a s hodnotou %d je v pameti ulozena na adrese %p", a, &a);
p = &a; // Uloží do p adresu proměnné a
*p = 15; // Uloží hodnotu 15 na adresu v p, to znamená že do proměnné a
printf("Promenna a ma hodnotu %d", a);
```

Program vypíše:

Promenna a s hodnotou 56 je v pameti ulozena na adrese 00ff12ab (například)
Promenna a ma hodnotu 15

Povolené operace ukazatelů:

int* + int = int*

*Posun v paměti (změna adresy paměti), když je přičtena hodnota n, adresa se zvětší (posune) o n*sizeof (int) – u int o „n“x 4 byty.*

Povolené operace ukazatelů:

int* + int = int*	*Posun v paměti (změna adresy paměti), když je přičtena hodnota n, adresa se zvětší (posune) o n*sizeof (int) – u int o „n“x 4 byty.*
int* - int = int*	*Posun v paměti (změna adresy paměti)*

Povolené operace ukazatelů:

int* + int = int*	*Posun v paměti (změna adresy paměti), když je přičtena hodnota n, adresa se zvětší (posune) o n*sizeof (int) – u int o „n“x 4 byty.*
int* - int = int*	*Posun v paměti (změna adresy paměti)*
int* - int* = int	*Vzdálenost ukazatelů (adres v paměti) Výsledek počet prvků int (4 byty) mezi adresami*

Povolené operace ukazatelů:

int* + int = int*	*Posun v paměti (změna adresy paměti), když je přičtena hodnota n, adresa se zvětší (posune) o n*sizeof (int) – u int o „n“x 4 byty.*
int* - int = int*	*Posun v paměti (změna adresy paměti)*
int* - int* = int	*Vzdálenost ukazatelů (adres v paměti) Výsledek počet prvků int (4 byty) mezi adresami*
int* relační operátor (<,>,!=,==) int* = int	*Porovná ukazatele (adresy paměti). Porovnání se provede odečtením adres. Když je výsledek 0, pak jsou stejné adresy. Jiný výsledek znamená, že nejsou stejné.*

Pointer (ukazatel) použijeme:

- když potřebujeme mít výstupní parametry ve funkci – parametry, které odchází z hlavního programu do funkce a funkce je může změnit (hodnoty proměnných) a vrátit do hlavního programu

Pointer (ukazatel) použijeme:

- když potřebujeme mít výstupní parametry ve funkci – parametry, které odchází z hlavního programu do funkce a funkce je může změnit (hodnoty proměnných) a vrátit do hlavního programu

- při dynamické alokaci paměti *(viz dále)*

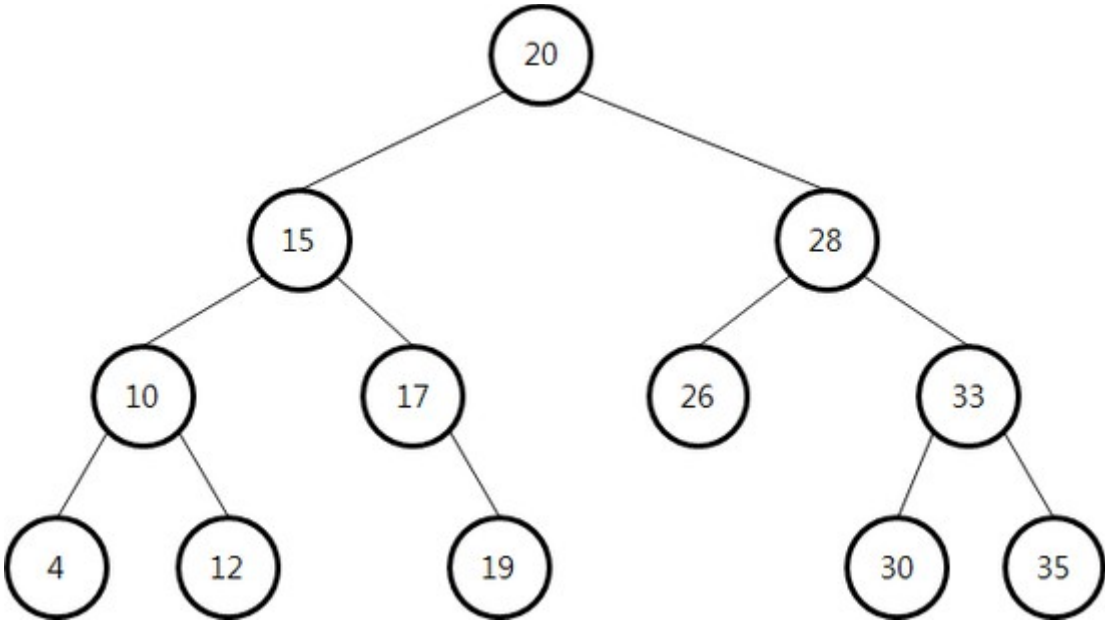
Pointer (ukazatel) použijeme:

- když potřebujeme mít výstupní parametry ve funkci – parametry, které odchází z hlavního programu do funkce a funkce je může změnit (hodnoty proměnných) a vrátit do hlavního programu

- při dynamické alokaci paměti *(viz dále)*

- *při složitějších datových strukturách (spojové seznamy, stromy)*

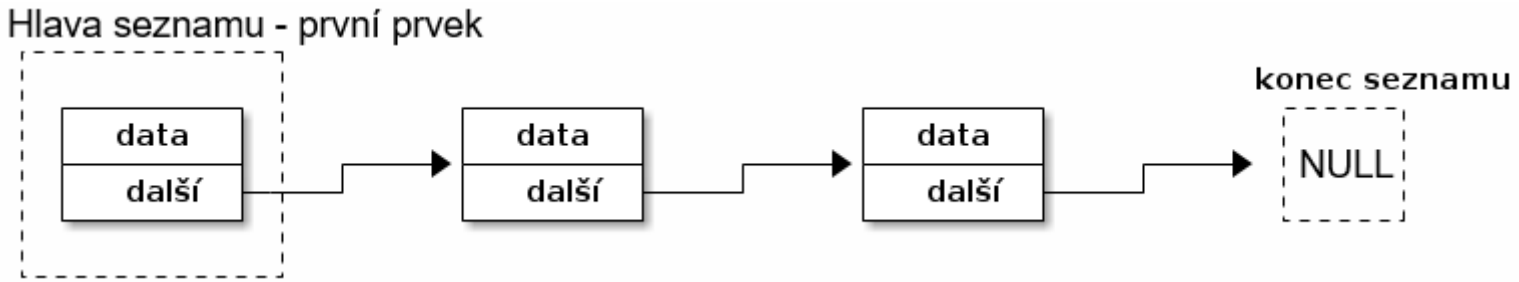
Binární vyhledávací strom:



Umožňuje rychlejší vyhledávání než pole. Nemusíme prohledat „n“ prvků, ale pouze dvojkový logaritmus „n“ : $\log_2(n)$.

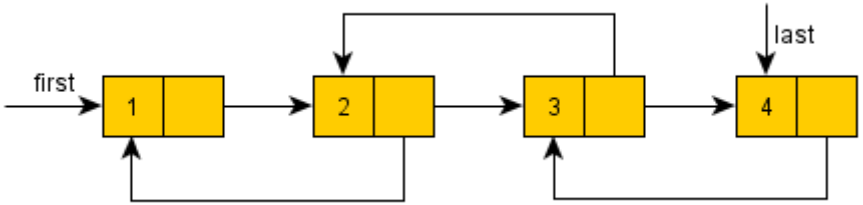
Pozor, každý prvek je v seznamu obsažen nejvýše jednou!

Jednosměrně zřetězený spojový seznam



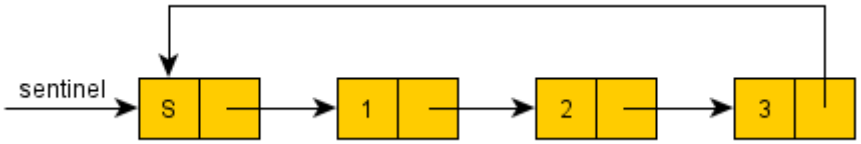
V prvku seznamu jsou data a odkaz na další prvek. Využívá se ve v datových typech fronta (údaj, který první přijde, první odejde) a zásobník (údaj, který první přijde, poslední odejde).

Obousměrně zřetězený spojový seznam



Každý prvek obsahuje odkaz na prvek následující a také předchozí.

Cyklicky zřetězený spojový seznam



sentinel je vstupní prvek

Každý prvek obsahuje odkaz na prvek následující, poslední prvek obsahuje odkaz na první.

Definice statické proměnné:

Je uložena **na zásobníku** (část paměti, která je přidělena programu – omezené místo řádově v MB), v proměnné je uložena přímo hodnota daného typu.

int A = hodnota; nebo ***int A;***	elementární statická proměnná
int A[10];	pole o deseti nedefinovaných prvcích
int A[] = {1,2,3,4,5,4,3,2,1,0};	automaticky se přiřadí počet prvků

U statického pole nelze měnit jeho velikost!

Definice dynamické proměnné:

Funkce pro dynamickou alokaci proměnné jsou v knihovně **stdlib**.

Je uložena na **haldě** (zbytek nevyužité paměti), v proměnné je uložen osmibytový (64 bit OS) nebo čtyřbytový (32 bit OS) pointer, tedy ukazatel na místo v paměti, na kterém je uložena hodnota. Výhodou je vyšší rychlost, oproti statickým proměnným, přiřazování hodnot k proměnným.

Definice dynamické proměnné:

Funkce pro dynamickou alokaci proměnné jsou v knihovně **stdlib**.

Je uložena na **haldě** (zbytek nevyužité paměti), v proměnné je uložen osmibytový (64 bit OS) nebo čtyřbytový (32 bit OS) pointer, tedy ukazatel na místo v paměti, na kterém je uložena hodnota. Výhodou je vyšší rychlost, oproti statickým proměnným, přiřazování hodnot k proměnným.

int *A = (int*)malloc(sizeof(int));
elementární dynamická proměnná s vyhrazením místa pro číslo typu integer bez přiřazení hodnoty

Definice dynamické proměnné:

Funkce pro dynamickou alokaci proměnné jsou v knihovně **stdlib**.

Je uložena na **haldě** (zbytek nevyužité paměti), v proměnné je uložen osmibytový (64 bit OS) nebo čtyřbytový (32 bit OS) pointer, tedy ukazatel na místo v paměti, na kterém je uložena hodnota. Výhodou je vyšší rychlost, oproti statickým proměnným, přiřazování hodnot k proměnným.

int *A = (int*)malloc(sizeof(int));
elementární dynamická proměnná s vyhrazením místa pro číslo typu integer bez přiřazení hodnoty

int *A=(int*)malloc(10*sizeof(int));
dynamické pole o deseti prvcích, první index 0, poslední 9

Definice dynamické proměnné:

Funkce pro dynamickou alokaci proměnné jsou v knihovně **stdlib**.

Je uložena na **haldě** (zbytek nevyužité paměti), v proměnné je uložen osmibytový (64 bit OS) nebo čtyřbytový (32 bit OS) pointer, tedy ukazatel na místo v paměti, na kterém je uložena hodnota. Výhodou je vyšší rychlost, oproti statickým proměnným, přiřazování hodnot k proměnným.

int *A = (int*)malloc(sizeof(int));
elementární dynamická proměnná s vyhrazením místa pro číslo typu integer bez přiřazení hodnoty

int *A=(int*)malloc(10*sizeof(int));
dynamické pole o deseti prvcích, první index 0, poslední 9

int *A=(int*)calloc(10, sizeof(int));
dynamické pole o deseti prvcích, první index 0, poslední 9, všechny prvky jsou vynulovány – mají hodnotu nula. *(pozor: namísto * je ,)*

malloc – memory allocation
malloc vrací hodnotu bez určení typu (void*), proto přetypujeme (int*)malloc

Změna velikosti pole:

A = realloc(A, 100*sizeof(int)) - z deseti prvků jsme změnili na 100 prvků,

realloc – reallocation

Změna velikosti pole:

A = realloc(A, 100*sizeof(int)) - z deseti prvků jsme změnili na 100 prvků,

realloc – reallocation

Není vhodné zvětšovat o malý počet prvků, protože zvětšení probíhá tak, že se nejprve naalokuje nová paměť, pak se prvek po prvku zkopíruje a nakonec se původní pole odstraní.

Změna velikosti pole:

A = realloc(A, 100*sizeof(int)) - z deseti prvků jsme změnili na 100 prvků,

realloc – reallocation

Není vhodné zvětšovat o malý počet prvků, protože zvětšení probíhá tak, že se nejprve naalokuje nová paměť, pak se prvek po prvku zkopíruje a nakonec se původní pole odstraní.

Dynamicky alokovanou proměnnou musíme, ve chvíli, kdy ji nepotřebujeme, vždy odstranit z paměti! Trvale by zabírala místo v paměti až do ukončení chodu programu, po ukončení chodu programu všechny proměnné použité v programu odstraní operační systém.

Uvolnění dynamické proměnné (typu pointer) z paměti (RAM)

free(A);

- uvolní prostor v operační paměti vyhrazený proměnné A, pozor: hodnota v paměti zůstává, pokud operační systém daný prostor nevyužije, je hodnota stále přístupná!

Uvolnění dynamické proměnné (typu pointer) z paměti (RAM)

free(A);

- *uvolní prostor v operační paměti vyhrazený proměnné A, pozor: hodnota v paměti zůstává, pokud operační systém daný prostor nevyužije, je hodnota stále přístupná!*

free(A);
A = NULL;

- *uvolní prostor v operační paměti vyhrazený proměnné A a do proměnné A vloží NULL, to znamená, že program ztratí adresu dat uložených v paměti, od té chvíle již nemáme přístup k datům*

Staticky definované dvourozměrné pole

int Pole [3][4];

[0][0]	[0][1]	[0][2]	[0][3]
[1][0]	[1][1]	[1][2]	[1][3]
[2][0]	[2][1]	[2][2]	[2][3]

Indexy v závorkách definují počet prvků y a x, nejvyšší index je o jedničku menší, protože první prvek má index 0 (nula)!

Dynamicky definované pole

```
int **Pole,i;
//deklarace ukazatele na ukazatel na integer
Pole=(int**)malloc(3*sizeof(int*));
for(i=0;i<3;i++) Pole[i]=(int*)malloc(4*sizeof(int));
```

Dynamicky definované pole

```
int **Pole,i;
//deklarace ukazatele na ukazatel na integer
Pole=(int**)malloc(3*sizeof(int*));
for(i=0;i<3;i++) Pole[i]=(int*)malloc(4*sizeof(int));
```

Dynamicky lze definovat pouze jednorozměrné pole. V případě potřeby definice vícerozměrného pole, např. dvourozměrného, alokujeme nejprve jednorozměrné pole ukazatelů, a potom pro každý ukazatel alokujeme pole integerů. Nejprve tak alokujeme adresy řádků a potom jednotlivé řádky.

Dynamicky definované pole

```
int **Pole,i;
//deklarace ukazatele na ukazatel na integer
Pole=(int**)malloc(3*sizeof(int*));
for(i=0;i<3;i++) Pole[i]=(int*)malloc(4*sizeof(int));
```

Dynamicky lze definovat pouze jednorozměrné pole. V případě potřeby definice vícerozměrného pole, např. dvourozměrného, alokujeme nejprve jednorozměrné pole ukazatelů, a potom pro každý ukazatel alokujeme pole integerů. Nejprve tak alokujeme adresy řádků a potom jednotlivé řádky.